University of Essex

DEPARTMENT OF ELECTRONIC SYSTEMS ENGINEERING

OPTICAL NETWORK DESIGN : VIRTUAL TOPOLOGY AND WAVELENGTH ASSIGNMENT USING LINEAR PROGRAMMING

DAMIANOS GAVALAS

MSc in Telecommunications & Information Systems

Supervisor : M.C.Sinclair

Essex, May 1997

DEPARTMENT OF ELECTRONIC SYSTEMS ENGINEERING

UNIVERSITY OF ESSEX

OPTICAL NETWORK DESIGN : VIRTUAL TOPOLOGY AND WAVELENGTH ASSIGNMENT USING LINEAR PROGRAMMING

AUTHOR : DAMIANOS GAVALAS

SUPERVISOR : M.C. SINCLAIR

This project has been submitted as a partial fulfilment of the conditions for the award of a MSc in Telecommunications & Information Systems

Essex, May 1997

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SUMMARY OF THE REPORT STRUCTURE

Chapter one seeks to introduce reader to the need for network design optimisation. There is also an overview of the problems that will be investigated in the chapters following, as well as references on COST 239 European Optical Network which has been widely used during the experimental part of the project work, in order to test the efficiency of optimisation techniques applied.

In *chapter two*, apart from some basic concepts of graph theory, there is presentation and comparison of the most well known formal optimisation techniques (Heuristics, Genetic Algorithms, Simulating Annealing and Linear / Integer Programming).

Chapter 3 deals with the first of the network design problems investigated in this report : “Virtual Topology Design” problem. The formulation of a Linear Programming (LP) problem corresponding to it is carried step-by-step. The construction of the cost function used is also described in detail, and the problem’s complexity is commented. Then, some results are presented and briefly discussed. The case of including backup paths in network topology is also examined, together with an effort to improve the optimisation results obtained.

In *chapter 4* there is an investigation of the second problem examined taking place : “Routing & Wavelength Assignment” problem. The formulation of the equivalent LP problem (objective function and constraints) is again described in detail. Initial results are presented and a further effort to obtain a solution that unthreads the limitations of LP is attempted.

In *chapter 5* there is a more detailed discussion of the results obtained, correlated to the limitations of LP. An Integer Programming (IP) approach of all the problems examined so far is also presented.

Finally, *chapter six* concludes the work carried out in this project and gives some directions for future work in the network design field. The remaining sections are the *acknowledgements*, the *references* and *appendices* to the work.

CHAPTER 1

INTRODUCTION

1.1. Network Design & Optimisation

Network Design has attracted raising attention during the last years. The increasing demand for capacity for the future unpredictable services and the resulting demand for high bandwidth which is driving new technologies into realisation, has contributed in the growth of interest in network design.

Network design problem includes various concepts and can be divided into the following sub-problems, many of which are currently widely investigated in the literature :

- Topological Design [3,7,13,14,19,20,28]
- Routing [13,14,16,17,25,26,29]
- Wavelength Assignment and Allocation (in optical networks) [5,17,22,23,26,27]

It should be noted that network design does not necessarily provide optimal or even near-optimal networks. However, in case that the objective is an optimal solution, network design issues have been proved complex enough to cope with manually, so there has been an effort to develop several formal ways to automate the procedure of *optimisation*. Thus, various optimisation problems have been arisen. Many different approaches have been applied to network optimisation problems, such as shortest path, link assignment, minimum delay, maximum flow, etc. Combinatorial optimisation is frequently used to find optimal or near-optimal solutions for similar problems.

Those optimisation problems are very complex. Thus, several methods have been proposed in order to be solved : Heuristics, Genetic Algorithms, Simulation Annealing, Linear & Integer Programming, etc. All these methods are discussed in the second chapter. A comparison of their performance has been also attempted.

1.2. Project Summary

This project mostly involves investigation of the potentials and the limitations of *Linear Programming* (LP) techniques for optical network optimisation purposes. The main motivation for the project has been to try LP on problems that had previously been tackled by using Genetic Algorithms [3,5].

Hence, LP has been applied in order to optimise various networks (COST 239 European Optical Network, presented in the following section, was one of them) and several results have been taken and discussed. The term "network optimisation" includes, in this project, the concepts of *optimised network topology* (as a first approach) and that of *optimised wavelength assignment* (as a second approach) in terms of *minimum cost.*

This means that a cost function, representing the cost of the network is minimised. The cost function is directly associated with the *objective function* of the Linear Programming (LP) problem. This function depends on some variables such as the link capacities as well as some constants (the link lengths, weighting factors, etc.). In addition to the cost function, the LP problem should also include some constraints which will be satisfied by the variables of the LP problem. Those variables may be included in the objective function, or they may be not.

In the case of the network virtual topology design, the aim is to find the optimum topology which minimises the cost of the network. In the other case, that of optimal wavelength assignment, the objective is to minimise, apart from the capacity of links, the number of channels (wavelengths) that are required to route the traffic load for each one of the network links. It should be noted that in this case, the network topology is fixed.

In both cases a LP-problem solving program is used, in order to give the optimal solution : *lp_solve_2.0* (it has been downloaded from an Internet site [44]). This program is based on the *simplex method* algorithm (see Appendix A) and has been developed by *Michel Berkelaar.* It is written in C programming language and, according to author statements, it has solved models with up to 30,000 variables and 50,000 constraints. However, it was proved that complex models caused this code problems, as the CPU time needed to solve problems was dramatically increased with complexity (commercial codes are more efficient, but they are quite expensive as well). Generally, lp_solve_2.0 is capable of solving *generalised LP* and *Mixed Integer Programming (MIP)* problems [41,42].

1.3. COST Action 239

"As the work of this project has been mostly implemented and tested on COST Action 239, an idea of the framework is deemed necessary. Also, a description of various design issues addressed in the COST 239 European Optical Network (EON) is made to give an account of recent developments of the Action" [4].

1.3.1. Framework

"The COST framework (European Cooperation in the field of Scientific and Technical Research) is run by the European Commission, aiming to promote pre-competitive R&D cooperation between industry, universities and national research centres.

COST 239, "Ultra-high Capacity Optical Transmission Networks", has studied the feasibility of an all-optical[1] network, capable of carrying all the international traffic between the main centres in Europe. At the start of the project, twenty such centres were identified, each acting as the gateway for all (or half) of the European international traffic for their country" [3,4,5,6].

Current studies on COST 239 are directed at investigating on the optical infrastructure [45] :

- the multiplexing techniques (Time Division Multiplexing, Wavelength Division Multiplexing, Optical Time Division Multiplexing)
- the high-capacity transmission techniques (linear and soliton propagation)

1.3.2. COST 239 European Optical Network

There are several issues under considerations in the development of an all optical network linking major centres in Europe. As the proposed network spans a large eographical area but with a relatively small number of nodes (20) much research work has now been actively undertaken to cater for it, taking into account not only the high

[1] A network is characterised as all-optical when no opto-electronic or electro-optical conversion takes place (both transmission and switching take place in the optical domain).

capacity transmission and network performance, but also the cost effectiveness of such technology that is to be implemented.

Several issues have been investigated, including the evaluation of traffic models for medium and long term demand, estimation of link capacity requirements, design of topology [3] to meet availability requirements such as reliability and blocking, evaluation of transmission requirements and limitations and evaluation of the number of wavelengths.

CHAPTER 2

LITERATURE SURVEY AND BACKGROUND

2.1. Modelling Networks as Graphs

2.1.1. Basic Graph Terminology

A *graph* is collection of a *vertex set V* and an *edge set E* [35,37,39].These vertices are more commonly called *nodes* and they represent locations (e.g. sources of traffic or sites of communications equipment). The edges are also called *links*, and they represent communication facilities. A link is incident on a node if the node is one of its endpoints. Nodes *i* and *k* are *adjacent* if there exists a link (i,k) between them. Such nodes are also referred to as *neighbours*. The *degree* of a node is the number of links incident on the node or, equivalently, the number of neighbours it has [33].

“A graph is called *network* if the links and nodes associated with it have properties (e.g. length, capacity, type, etc.). A *path* on the network is a sequence of links which begins at some node, *s*, and ends at some node, *t*. Such a path is also referred to as *s,t-path*. It should be noted that the order of links in the path matters. In a communications network a channel is said to be *full duplex* if it can be used in both directions at the same time and *half duplex* if it can be used in only one direction at a time” [31].

2.1.2. Internal Representation of Networks

The internal representation of the network is aimed at making the network design and analysis algorithm easy to implement and efficient both in terms of their memory requirements and running time.

2.1.3. Node and Link Numbers

“Nodes are identified by numbers; conceptually, each node has a number associated with it. This number may be its position in an alphabetical list of node ID, an index into an array of node properties, or a pointer to a record describing the node. In all cases, this number provides immediate access to information about the node. Thus, the first step in creating the internal representation, is to associate a number with each node.

In a similar way, numbers can be associated with the links and information about the links accessed via these numbers. The endpoints of the links are node numbers. If the node numbering is changed, the link endpoints must also be converted" [31].

2.1.4. Adjacency and Incidence Relations

"Many algorithms make use of data structures that indicate which nodes are adjacent to others and which links are incident on each node. There are several common ways of representing this information. One of the simplest ways is for a network with N nodes, to create an N by N boolean matrix with 1s, where nodes are adjacent, and 0s elsewhere. Similarly, for a network with N nodes and M links, it is possible to create an N by M matrix with 1s in rows i and j of column k, if undirected link k connects nodes i and j. It should be noticed that the adjacency matrix is always symmetric, in contrast with the incidence matrix which is not necessarily.

A matrix is said to be *sparse* if the majority of its entries are 0s, namely if the number of non-zero entries is very low comparing to the size of the matrix. Finally, the degree of a node can be computed by adding up the number of 1s in the row of the incidence or adjacency matrix corresponding to that node" [31].

2.2. Optimisation Techniques

2.2.1. Heuristics

"Heuristics are design principles that are incorporated into algorithms. They are good ideas embodying design experience. The intuition behind the heuristics is to start with simple rules to investigate how they perform and use the experience gained to implement more advanced and complicated rules. As a result, the application of simple heuristics may guide us to even better ones. Although they do not guarantee an optimal solution, they can provide an acceptable solution to problems that are hard to compute within reasonable time and effort" [7].

Some heuristics are specific to particular types of networks. The most useful ones, are based on principles which can be applied across many types of networks. One of the most widely used heuristics is the *greedy algorithm*. Confronted by a series of choices, the greedy algorithm chooses the best one it can find at each stage.

“The greedy algorithm is a broadly applicable heuristic based on the simple observation that inexpensive networks tend to contain inexpensive links. This principle is true for all types of networks. Therefore this algorithm yields a “good” network, but not necessarily the optimal solution. It is quite possible that the least expensive network does not contain some of the least expensive links. It happens that in the case of this simple unconstrained network, the greedy algorithm does in fact yield the optimal solution. However, with added constraints, the greedy algorithm no longer yields the optimal solution” [31].

Heuristics based Network Design in Literature

Banerjee, Mukherjee and Sarkar presented a comprehensive paper on heuristics for linear multihop lightwave networks [12], in which a greedy and an iterative approach are applied to two different sets. The first set focuses on *minimising the maximum flow* in any link of the network, while the second set *minimises the average delay*, given the traffic and the distance matrices. Another heuristic algorithm was presented by Kershenbaum, Kermani and Grover [13] dealing with the problem of obtaining a minimum cost topology for a mesh network, given matrices specifying the traffic and the cost of links between all node pairs.

Gerla and Kleinrock [28] suggest a heuristic topological design procedure as a *solution to the problem of data transmission in a network environment*. Various problem formulations have been considered, including the minimisation of the total line cost, the communication cost or the source to destination packet delay as objectives.

Numerous heuristics have been applied to the *wavelength assignment problem* as well. Wauters and Demeester [21] considered separately the case where an optical path has a fixed wavelength and is denoted a Wavelength Path (WP), and the case where wavelength translating cross-connects are used and an optical path can have different wavelength on each fibre (Virtual Wavelength Path : VWP). They introduced an heuristic trying both to *maximise the wavelength reuse and minimise the network wavelength requirement*.

Finally, Wuttisittikulkij and O’Mahony [27] developed a *simple heuristic algorithm for wavelength assignment in optical networks*. The objective is to assign wavelengths to all connections in such a way that the number of wavelength channels required, M, is minimal. The main issue that this paper addresses is the problem that arises when some additional connections are to be established or the traffic between some node pairs is to be increased in an already optimised network. In that case, the presented algorithm would have the great advantage of *not requiring any modification of the existing connections*.

Hence, all these heuristics based attempts for network design optimisation converge to the fact that heuristics can be an effective, non time demanding tool providing sufficient solutions, although they cannot guarantee optimality.

2.2.2. Genetic Algorithms

The term *genetic algorithms* (*GAs*) describes the properties of a class of evolutionary algorithms (*EAs*) in the artificial system. EAs use computational models of evolutionary processes as key elements in the design and implementation of computer-based problem-solving systems [1].

GAs can be distinguished from EAs in general by two key-features :

- They operate in a *search space* using genetic operators on fixed-length strings that *encode* points in the *problem space*, rather than in the problem space directly.
- They use a particular recombination operator on these fixed-length strings that is unique to GAs, called *crossover*.

GAs are usually used to deal with problems of optimisation as well as learning problems. Apart from that, GAs have also been used to design application-specific neural networks. In the communication field, GAs have been used to optimise the design of communication networks.

GAs based Network Design in Literature

GAs are frequently used in network design problems in order to provide optimal or near-optimal solutions.

Sinclair [3] has applied GAs in the optimisation of the COST 239 EON topology. The author has shown that GAs have not only successfully optimised the cost of the network but also increased its reliability. Tan and Sinclair [5] presented complementary work for wavelength and routing assignment between the central nodes of the same network.

Hewitt, Soper, and McKenzie [9] presented a GA for mesh network design, considering both capacity and wavelength allocation. Their results were comparable with existing heuristics, and often better in terms of overall network cost.

Hence, GAs obviously concentrate all the advantages of heuristics providing satisfactory solutions in a finite time duration and, sometimes giving better results comparing to them.

2.2.3. Simulated Annealing

"Simulated Annealing (SA) is a process whereby candidate solutions to a problem are repeatedly evaluated according to some objective function and incrementally changed to achieve better solutions. The nature of each individual change is probabilistic in that there is some probability it worsens the solution. However, the probability is skewed so that changes which serve to minimise the objective function occur with higher frequency. In addition, an "annealing schedule" is followed whereby the probability of allowing a change which worsens the solution is gradually reduced to zero" [22].

Thus, SA is similar to gradient decent algorithms, wherein each change to the candidate solution is made such that the objective function is always minimised. The fundamental difference is that SA assumes no initial knowledge of the gradient. The fact that SA algorithms occasionally accept deteriorations in cost in a controlled manner, enables them to escape from local minima while keeping the favourable features of local search algorithms, i.e., simplicity and general applicability [23].

SA based Network Design in Literature

Ganz, Gong and Wang [22] used a SR[2] optimisation algorithm to approach the wavelength assignment in multihop lightwave networks problem. The aims considered in their paper, were to increase the user accessible bandwidth, reduce the average delay and reduce the hardware cost (in terms of the number of transmitters and receivers per node and numbers of wavelengths in the network).

[2] Stochastic Ruler : A method similar to SA

2.2.4. Linear / Integer Programming

Except for the problems with small complexity, it is not possible to enumerate all possible solutions and then choose the best one. A network with ten nodes, for example, has 45 potential links[3], each of which could be either included or excluded from the network. Even if it is assumed there is only one possible link, this gives rise to 2^{45} possible solutions (more than 10^{13} possibilities). The set of all possible solutions to a problem is referred to as the *solution space* or *problem space*.

“The notion of best is expressed in formal optimisation techniques by the *objective function*. The objective function associates a value with the design variables. Thus, for example, each link *i*, can be included or excluded from a design. There is a cost associated with each link and in order to minimise cost, the sum of the costs of the links chosen should be used as the objective function.

While cost is the most commonly used objective function, it is not the only one. We might want to maximise reliability. In this case, associate reliability with each possible node and link in the network, and then compute the reliability of candidate networks comprised of specific nodes and links. Unlike the case of cost, however, this may be a complex calculation in its own right. The best value of the objective function is referred to as the *optimum*. As it can be seen from the two previous examples, the optimum can be either a minimum or a maximum.

If the optimum is to be found for problems of realistic size, we must rely on properties of the problem which allow us to avoid at the most of the possible solutions. One such situation was mentioned before; the greedy algorithm, which examines only a very limited number of alternatives and guarantees an optimal solution for a special class of problems. There are also other situations where a general heuristic produces optimal solutions.

There are also algorithms which always produce optimal solutions. The problem with these, however, is that they only apply to a limited class of problems; for problems outside this class, they do not work at all. One such algorithm is the *simplex method*. This algorithm only works for the class of problems called ***linear programming problems***. Linear programs are problems where both the constraints and the objectives are weighted sums of the variables; that is, they are of the form :

[3] In general, a network with N nodes can have a maximum of N * (N-1) / 2 links

$$z = \sum_{i=1}^{N} a_i x_i$$

where the x_i are the N variables that we are optimising over, and the a_i are the respective coefficients" [31].

In this case, the solution space can be searched in a very orderly way, avoiding most of the possible solutions and reaching the optimal solution in a reasonable amount of time. Linear Programming concepts are discussed in more detail in Appendix A, together with a description of the Simplex Method.

In literature (textbooks), there is a variety of problems which are coped with LP approach : Primal Cost Improvement [39], Maximal Flow Problem [37, 41], Transportation Problem [40], Assignment Problem [37, 40], Shortest Path Routing [33, 37, 39], Capacity Assignment Problem [32, 39], Minimum Cost Flow Problem [39, 41], Maximal Stiffness of a Structure [14], Optimal Planning of Transmission Network Investments [15], etc.

Often, a problem can be phrased as a linear program with the additional constraint that the variables must be integers. These problems (called ***integer programming problems***) are much more difficult to solve than linear programming problems. In particular, the simplex method and its variants do not work on integer programming problems. It should, however, be mentioned that integer programming problems are much more time consuming, in terms of the CPU time they need to be solved, compared not only to the equivalent linear programming problems, but all the other formal methods of optimisation as well.

Integer programming problems arise frequently in network design; that is, when, for example, the decision is whether or not to include or not a link in the solution. The choice is to include the link ($x_i = 1$) or not to include ($x_i = 0$); there is no middle ground. This type of integer programming problem are called ***zero-one programming problem*** [37,38].

LP/IP based Network Design in Literature

LP/IP techniques are widely used in literature to cope with network design optimisation problems. However, as it will be clear by the paragraphs following, IP as well as Integer-Linear Programming (ILP) are much more frequently used, comparing to LP, as they have been proved more efficient methods to deal with this kind of problems.

IP has been recently applied to the wavelength assignment problem. It serves as a comparison with other methods (e.g. heuristics) and it also provides lower and upper bounds to the problem. Ramaswami and Sivarajan presented a Routing and Wavelength Assignment (RWA) algorithm [17] which meets an upper bound on the carried traffic of connections (or, equivalently, a lower bound on the blocking probability). The RWA problem is formulated as an ILP [37,41,42] problem where the objective is to maximise the number of connections that are successfully routed. The outcome of an RWA algorithm can be represented by a path-wavelength assignment matrix, where each element is 1 if the algorithm assigns a wavelength to a connection and 0, otherwise. The ILP must maximise the carried traffic, and also satisfy the constraint that the same wavelength is used at most once on a given link.

Lowe, Shepherd and Walker [16] investigated the performance of two tools for path routing and wavelength allocation : a heuristic algorithm based on the idea of maximising reuse of fibres as further path requests are satisfied, and an IP formulation configured to minimise the total number of fibres used in the network. The two approaches are complementary, in that the heuristic algorithm runs quickly, whereas the IP approach yields lower bounds which guarantee optimality.

Wauters and Demeester developed an ILP model to deal with the problem routing and wavelength assignment in WDM optical networks, with or without wavelength conversion [25]. Routing cost is introduced in the objective function by using a weight function which favours the use of shortest paths.

Banerjee and Mukherjee [26] approach the problem of Routing and Wavelength Assignment (RWA) in large WDM optical networks. They develop an ILP formulation with the objective function being to minimise the flow in each link (their main aim is to minimise the number of wavelengths needed to carry a certain number of connections, assuming a known physical topology). They consider at most one lightpath from a source to a destination, so the variables denoting the corresponding traffic (in terms of a lightpath) have only 1 and 0 as valid values.

Dutta [18] has developed an IP model to determine transmission capacity in private backbone networks. Variables representing the cost of installing lines and decision variables (set to 1 if a line of specific type is installed, to 0 otherwise) are included in the model formulation. A similar approach (usage of decision variables) is proposed in Chapter 5, where the formulation of an IP problem, that would give an optimal solution for the two network design problems under investigation (virtual topology and wavelength assignment), is attempted.

Summarising, it could be stated that the enormous time requirements of LP/IP/ILP techniques make them usable only in case that providing an optimal (and not a near optimal) solution is of extreme importance, or in case that the effectiveness of another optimisation method has to be measured, in other words, when lower and upper bounds have to be provided.

2.2.5. Comparison of Optimisation Techniques

As it has been already clear by the brief description of the optimisation techniques preceded, there is not any method that could be characterised as a perfect one.

Heuristics have the advantage of *simplicity* and *speed* (they need a small amount of time to solve the problem). However, the solution obtained is "*near optimal*", namely they do not always provide the best solution possible.

Genetic Algorithms present the great advantage of their *probabilistic nature*, that make them ideal for problems of optimisation as well as learning problems. On the other hand, when *many iterations* are needed, they are quite *demanding in terms of time* needed in order to give the solution (there is *dependence on the problem space*), which is *not necessarily the best*.

Simulating Annealing algorithms maintain the *simplicity*, *applicability* and *speed* of local search algorithms and the *probabilistic nature* of GAs while being capable of *escaping from stacking in local minima*. Nevertheless, they have all the disadvantages associated with the random-search methods which *cannot secure the best solution*.

Linear Programming presents the advantage of an exhaustive-search method, as it *always provides the best solution possible*, provided that the constraints have been correctly set. However, apart from the disadvantage of *solving only problems in strictly linear form* (both the constraints and the objectives are weighted sums of the variables), LP is *depended on the size of the problem space* : the number of variables is dramatically increased with the increase of problem space.

Integer Programming *includes all the advantages of LP*, together with the fact that the *linearity requirement no longer exists* (it can be required that some variables should take only integer values). The main disadvantage for IP is, as it is the case for LP, that it takes an *extremely large amount of time* (days !) to give a feasible solution.

Thus it can be concluded that the selection of the technique that will be applied in order to solve an optimisation problem, depends completely on the nature of the problem : whether the problem space is large or not, whether a really optimal solution is desired, etc. For example, methods such as GAs, although they cannot guarantee the best result, may still provide useful results for problems that are too large for LP or IP.

2.3. Cost Models

The basic idea in the case of topological design is to optimise a specific *cost function* of the network. The feasibility of the solutions can be controlled by appropriate penalty functions in case that heuristics or GAs techniques are applied [3,7].

The cost model used plays a crucial role in the network design. However, it is extremely hard to estimate and model the cost of a network, as it depends in a variety of factors. For example, the cost of the fibres and all the components required, apart from the cost of maintenance, must be taken into account. A simple and quite reasonable way to deal with this problem is to consider the cost as a linear function of length and capacity. The cost model mainly used in this project is a modification of the one described in [2] (it is simpler than this).

2.4. WDM / OTDM Optical Networks

Wavelength Division Multiplexing (WDM) is an emerging technology to enhance the capacity of the existing optical communication systems. In WDM networks, links consist of fibre mediums, while nodes contain switching elements with lasers (as transmitters) and photodetectors (as receivers). In these networks, nodes are interconnected by optical fibre highways carrying wavelength multiplexed optical channels. At a wavelength dependent cross-connect located at a network node, the wavelengths are demultiplexed and cross-connected to construct the network topology.

Multi-wavelength transparent optical networks are capable of providing high aggregate capacities and could provide an infrastructure which could overlay an existing transport network to meet future increases in traffic demand as well as to enhance flexibility and reliability [10].

WDM techniques combined with optical amplifiers [43] has made the concept of large scale WDM optical networks such as the COST 239 European Optical Network possible. The WDM technique is used to enhance cable capacity by the simultaneous transmission of a number of signals on different wavelengths in the same fibre. This technique allows the vast bandwidth available in an optical fibre to be utilised by partitioning it into several channels, each at a different optical wavelength. This has several advantages over existing transmission techniques. Rapid research progress in WDM technology has enabled such technology to become practical enough to expand their use over the last decade. However, these fields are beyond the scope of this report and only necessary background is provided.

WDM networks currently detain a great part of the literature related to optical networks, world-wide. For example, Chlamtac, Farago and Zhang [29] dealt with the problem of efficient circuit switching in WANs. Wavelength conversion has been considered in networks junctions. The authors have applied an interesting cost structure of using the resources. They assign costs not only for using a wavelength in a specific link but for wavelength conversion from a wavelength to another at each node. The wavelength conversion cost is dynamically adjusted, depending in network conditions (for example, when the traffic going through a node is getting closer to the node capacity, the wavelength conversion cost at this node is increased and the connections are routed through other nodes).

In the near future, the construction of large-capacity flexible optical networks, using both Optical Time-Division Multiplexing (OTDM) [11] and WDM technologies, will be of vital importance. A significant part of the research carried out in the area of optical networks deals with the combination of these two multiplexing techniques.

For example, a low-noise supercontinuum (SC) broadband source with a bandwidth of more than 200 nm (25 THz) has recently been developed and has applied to 200 Gbps TDM as well as 100 Gbps x 4 channel OTDM/WDM transmission [30].

CHAPTER 3

VIRTUAL TOPOLOGY DESIGN PROBLEM

3.1. Introduction

In this approach, the names and the geographical locations of the network switching nodes are given as input. In addition to these, the traffic matrix (a matrix that contains the units of traffic for each one of the node pairs), is also given. The final aim is to find the topology that gives the minimum possible cost. For simplicity reasons, the *cost of network* is assumed to be consisted of the *nodes* plus the *links cost*.

It should be noted that, initially, a *complete meshed* network (there are links connecting every one of the node pairs) is considered. The objective is to obtain a final topology that satisfies the requirement of minimum network cost. That means that some of the variables representing the links capacities have to be assigned a zero value. In case that a specific link has zero capacity, the traffic between the nodes that this link would connect, will be routed through another route (that will be a series of other existing links).

Of course, this depends, to a great extend, on the routing algorithm used. If, for example, a network with strictly direct-path routing is considered , it is obvious that all the links of the fully-meshed network are going to be used. This is because the traffic with source node A and destination node B will have to be routed through the link that connects nodes A and B, otherwise the direct-path requirement will not be satisfied.

3.2. Formulation of the Problem

3.2.1. Objective Function

In order to formulate the topology optimisation procedure as a LP problem, there should be a symbolic representation of the constituting elements of the network with some LP variables. Thus, we define some variables : x_i, *$i = 0,1,2$ ... Number of Links*, representing the capacity of links. There will follow the definition of some new variables representing the valid network paths, according to the routing algorithm used.

In that case, the cost of a link will be the product of its capacity multiplied its corresponding length. Thus, the cost of network links will be equal to the sum of the individual link costs and will be given by (3.2-1). It should be mentioned that for a network with N nodes, the maximum possible number of links (the number of links when the network is fully meshed), is : *N * (N - 1) / 2*.

$$C_L = \sum_{i=0}^{NumOf\ Links} x_i * L_i \qquad (3.2\text{-}1)$$

where x_i is the capacity and L_i the length of the i-link. The dependence of link cost on its capacity and length is quite reasonable, as in the real world, the more the capacity and the length of a link, the more it costs to be fabricated. The way that the lengths L_i are calculated will be discussed later.

On the other hand, the cost of a network switching node is considered to be equal to the sum of capacities of the links attached to it, multiplied with a factor A (the factor is introduced in order to provide a balance between the links and nodes cost in the network). As previously, the cost of network nodes will be equal to the sum of the individual nodes costs and will be given by :

$$C_N = \sum_{i=0}^{NumOf\ Nodes} N_i \qquad (3.2\text{-}2)$$

where N_i is the cost of the i-th node, given by :

$$N_i = A * \sum_{j=0}^{NumOf\ Attached\ Links} x_j \qquad (3.2\text{-}3)$$

For simplicity reasons and as this wouldn't cause any change in the results obtained, the factor A will be considered equal to unity : $\underline{A = 1}$.

The dependence of the node cost on the number of attached links is also realistic, as the complexity and, thus, the cost of every real switching node is associated with the number of its input and output ports. Finally, the total cost of the network is the sum of links and nodes cost :

$$C_{Net} = C_L + C_N \qquad (3.2\text{-}4)$$

However, the notation giving the total nodes cost can be significantly simplified, by examining carefully the following example. In the simple 5-nodes network of figure

3.1, the cost of each one of the 5 nodes can be found by simply summing up the capacities of the attached links :

$$
\begin{aligned}
N_0 &= x_0 + x_1 + x_2 + x_3 \\
N_1 &= x_0 + x_4 + x_5 + x_6 \\
N_2 &= x_1 + x_4 + x_7 + x_8 \\
N_3 &= x_2 + x_5 + x_7 + x_9 \\
N_4 &= x_3 + x_6 + x_8 + x_9
\end{aligned}
\qquad (3.2\text{-}5)
$$

The total nodes cost can be obtained by summing the previous set of equations :

$$C_N = \sum_{i=0}^{4} N_i = 2*x_0 + 2*x_1 + 2*x_2 +2*x_3 +2*x_4 + 2*x_5 +2*x_6 +2*x_7 +2*x_8 +2*x_9 =$$

$$= \sum_{i=0}^{9} 2x_i \qquad (3.2\text{-}6)$$

Thus, it is obvious that in any case of a fully meshed network, the total cost of network nodes would be given by the sum of the links capacities, multiplied with a factor of 2.

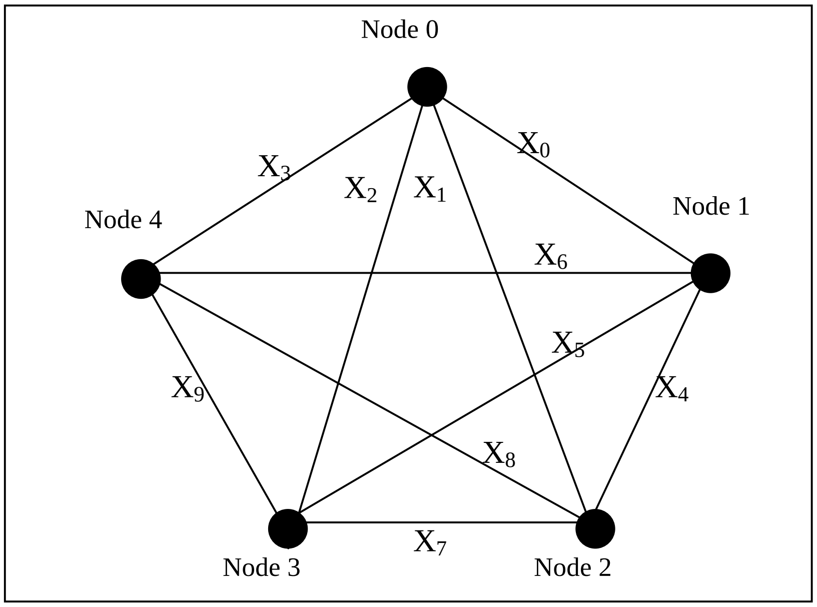


Fig. 3.1 : A fully meshed 5-nodes network

The combination of equations (3.2-1), (3.2-4) and (3.2-6) gives the final equation that describes the cost of the network :

$$C_{Net} = \sum_{i=0}^{NumOf\ Links} (2 + L_i) * x_i \qquad (3.2\text{-}7)$$

The previous equation, apart from calculating the network cost, is also used as the *objective function* of the LP problem. The dependence of the network cost on the links length is supposed to be used in order to force the LP executing program to give a solution, in which <u>no</u> links with length much greater to the average length will be used in the final optimum topology.

Concerning the calculation of the lengths $L_{i,}$ it has been previously mentioned that the names and the locations of the switching nodes are given as input. Hence, as we have the location (co-ordinates : x, y) each one of the nodes, we can evaluate the *Euclidean Distance* between them. If, for example, the location of node A is described by the co-ordinates (x_1 , y_1), and the equivalent co-ordinates of node B are (x_2 , y_2), the distance between A and B is :

$$d\,(A,B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \qquad (3.2\text{-}8)$$

That way, we evaluate the distance between each node pair.

3.2.2. Link Constraints

Apart from the objective function, the LP problem needs some constraints concerning the links, in order to define the solution area and, finally, the optimal solution vector (the value of each one of the variables used) which will minimise the objective function. There are two kind of links constraints :

♦ <u>*Constraint on "cuts around nodes"*</u> : The links attached to each node are supposed to have total capacity larger or equal to the sum of traffic units with source or destination that node. That means that there will be a number of inequalities of this kind, *equal to the number of nodes* [46].

Let's consider the network of fig. 3.1, and assume that the traffic matrix[4] has the form given by the table following :

[4] The Traffic Matrix consists of an m x m array, with empty forward diagonal, of the capacity required (in traffic units) between each node pair

Nodes	**0**	**1**	**2**	**3**	**4**
0	-	2	1	3.5	1.5
1	3	-	4	1	1
2	5	3.2	-	7	2
3	4	1	1.5	-	3
4	2	4	0	2.5	-

Table 3.1 : Traffic Matrix of the 5-nodes network

As it is clear, from the table above, there are : 2 + 1 + 3.5 + 1.5 = 8 traffic units with *source* the Node 0, and destination the other nodes and : 3 + 5 + 4 +2 = 14 traffic units with *destination* the Node 0 and source the other nodes. Thus, the links attached to Node 0 should be able to carry the traffic generated or directed to this node, so their *total capacity must be greater or equal to* : 8 + 14 = *22 traffic units*.

Likewise, there are other 4 inequalities, each of which will set the constraint for the remaining 4 nodes. These inequalities (including the one corresponding to Node 0) will be :

Node 0 : $x_0 + x_1 + x_2 + x_3 >= 22$

:

Node 4 : $x_3 + x_6 + x_8 + x_9 >= 16$

♦ *Constraint on "cuts around links"* : An another constraint is added if we remove each one of the network links that connects a specific node pair, and require the rest of the links attached to these two nodes to have total capacity greater or equal to the sum of traffic units with source or destination the two nodes, apart from the traffic units with source one of the two nodes and destination the other. Namely, there will be a number of inequalities of this kind, *equal to the number of links* : N * (N - 1) / 2, where N is the number of nodes [46].

Thus, for the network of fig.3.1 and for the traffic matrix given by table 3.1, if we remove the link that connects Node 0 with Node 1, then the rest of the links attached to these two nodes (x_1, x_2, x_3 and x_4, x_5, x_6 respectively) should be able to carry the traffic with source or destination Nodes 0 and 1, apart from the traffic with source Node 0 and destination Node 1, and inversely. This is clearly depicted in fig.3.2.

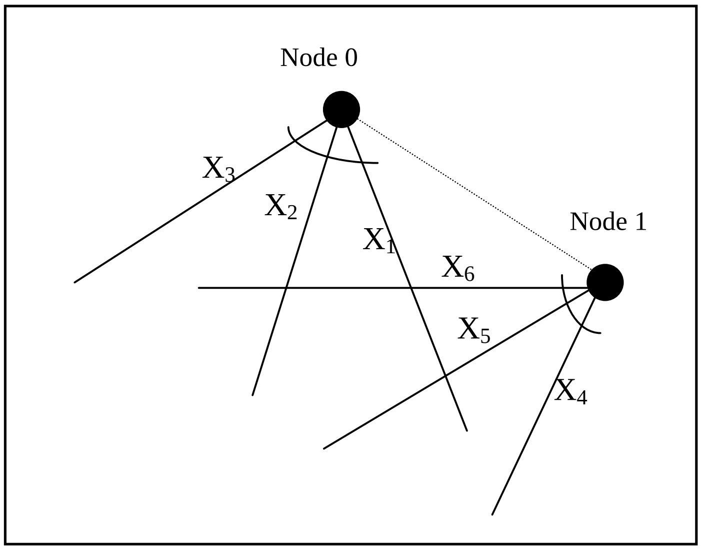


Fig. 3.2 : Cut around the link that connects nodes 0 and 1

The constraints are set similarly for the other links. Thus, we have the 5 * 4 / 2 = 10 inequalities :

Link 0 :	$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 >= 31.2$
Link 1 :	$x_0 + x_2 + x_3 + x_4 + x_7 + x_8 >= 33.7$
:	
Link 9 :	$x_2 + x_3 + x_5 + x_6 + x_7 + x_8 >= 28.5$

3.2.3. Path Constraints

As it happened with the links case, there are two kind of links constraints concerning the paths variables y_i :

♦ *Constraint on the sum of paths variables corresponding to a specific node pair* : For each pair of nodes, there is a variable y_i assigned to the direct, the 2, 3, ... -hop paths, depending on the number of hops used by the routing algorithm. Each of the variables represent the amount of traffic (a fraction of the total traffic between the two nodes) carried by the corresponding stream. Consequently, there is a requirement for the sum of traffic units carried by those streams to be equal to the traffic between the two nodes. This constraint produces a set of equalities, *equal to the number of node pairs*.

Of course, in a fully meshed network, the number of node pairs is identical to the number of links [46].

If, for example, a 2-hop routing algorithm is implemented in the fig. 3.1 simple network, we would have the following path variables, corresponding to the Node 0 - Node 1 node pair :

– Node 0 - Node 1	:	y_0
– Node 0 - Node 2 - Node 1	:	y_1
– Node 0 - Node 3 - Node 1	:	y_2
– Node 0 - Node 4 - Node 1	:	y_3

In this example, the sum of these variables should give the traffic between nodes 0 and 1, which according to the traffic matrix will be :

$$y_0 + y_1 + y_2 + y_3 = 2 + 3 = 5$$

The problem of evaluation of the number of hops (direct, 2-hop, 3-hop, etc.) that arises is discussed in detail in *Appendix B.*

♦ *Constraint on the sum of paths variables sharing the same link* : It is clear that a link might be shared between many distinctive streams, corresponding to different paths. Thus, it is required that the sum of those streams must be equal to the link capacity. The number of equalities generated as a consequence of this constraint will be *equal to the number of links* [46].

In the 5-nodes network, for example, the link x_0 (that connects the nodes 0 and 1), can be used by the paths :

– Node 0 - Node 1	:	y_0
– Node 0 - Node 1 - Node 2	:	y_5
– Node 0 - Node 1 - Node 3	:	y_9
– Node 0 - Node 1 - Node 4	:	y_{13}
– Node 1 - Node 0 - Node 2	:	y_{17}
– Node 1 - Node 0 - Node 3	:	y_{21}
– Node 1 - Node 0 - Node 4	:	y_{25}

Thus, the sum of those variables must not exceed the capacity of the link that connects the nodes 0 and 1 (x_0) :

$$y_0 + y_5 + y_9 + y_{13} + y_{17} + y_{21} + y_{25} = x_0$$

3.2.4. Synopsis of the LP problem formulation

As it is apparent, from the discussion above, the form of the LP problem of topology optimisation, will be the following :

$$\min : C_{Net} = \sum_{i=0}^{NumOf\ Links} (2 + L_i) * x_i \quad : \textit{Objective Function}$$

$$\left\{\begin{array}{l} x0 + x1 + x2 + x3 >= 22 \\ x0 + x4 + x5 + x6 >= 19.2 \\ : \\ : \end{array}\right\} : \text{Num of Nodes inequalities}$$

$$\left\{\begin{array}{l} x1 + x2 + x3 + x4 + x5 + x6 >= 31.2 \\ x0 + x2 + x3 + x4 + x7 + x8 >= 33.7 \\ : \\ : \end{array}\right\} : \text{Num of Links inequalities}$$

$$\left\{\begin{array}{l} y0 + y1 + y2 + y3 = 5 \\ y4 + y5 + y6 + y7 = 6 \\ : \\ : \end{array}\right\} : \text{Num of Node Pairs equalities}$$

$$\left\{\begin{array}{l} y0 + y5 + y9 + y13 + y17 + y21 + y25 = x0 \\ y1 + y4 + y10 + y14 + y17 + y29 + y33 = x1 \\ : \\ : \end{array}\right\} : \text{Num of Links equalities}$$

The aim is to find a solution vector (the values of all x_i and y_i) that minimises the objective function while satisfying all the constraints posed.

It can be shown that the cost of paths (the cost of a path is similarly defined as the capacity of the corresponding stream multiplied with the path total length) is equal to the cost of links. Thus, the objective function of the LP problem given above, can be substituted so that it includes the paths cost instead of the links cost. In that case, the total network cost (nodes cost plus the paths cost) would be :

$$C_{Net} = \sum_{i=0}^{NumOf\ Links} 2x_i + \sum_{j=0}^{NumOf\ Paths} y_j L_j \qquad (3.2\text{-}9)$$

where L_j is the length of the j-path, equal to the sum of lengths of the links it is consisted of. Therefore, in case that 2-hop routing is chosen for a 5-nodes network, the previous equation (objective function) can be written :

$$C_{Net} = \sum_{i=0}^{NumOf\ Links} 2x_i + y_0 L_{01} + y_1 L_{021} + y_2 L_{031} + y_3 L_{041} + y_4 L_{02} + \ldots =$$

$$(3.2\text{-}10)$$

$$= \sum_{i=0}^{NumOf\ Links} 2x_i + y_0 L_{01} + y_1 (L_{02} + L_{21}) + y_1 (L_{03} + L_{31})\ y_1 + (L_{04} + L_{41}) + y_4 L_{02} + \ldots$$

Obviously, the total length of a 2-hop path would be, in any case, greater than the length of the equivalent direct path. That would have an effect on the cost of network : according to the equation giving the objective function of the LP problem, the paths cost would be increased in case that, a 2-hop path would be selected for routing the traffic of a specific pair of nodes, instead of the direct path.

However, this logic would not be valid in a real network. In real networks there is another important factor that should be taken seriously into consideration. That is the *cost of establishment of a link* which would be one of the dominants factors of cost. This cost can be justified not only by the cost of installing a new connection, placing the cables etc., but by the increased cost of switching nodes as well (obviously, the larger the number of links attached to a switch, the more complex and expensive the switch is). Therefore, in some cases, when we deal with a pair of nodes which are located in distant places, and there are not significant bandwidth requirements for the traffic between them, then it seems that it doesn't worth to install a new link that connects directly these two nodes, especially if there can be found a path with total length not significantly greater than the length of the direct link.

3.3. Problem Complexity

It should be stated that there is direct relationship between the number of variables used and the problem complexity. That relationship is graphically depicted in figures 3.3 and 3.4. The former illustrates the dependence of the number of variables of the LP problem on the number of network nodes, whereas the later presents the number of variables as a function of the number of hops used by the routing algorithm.

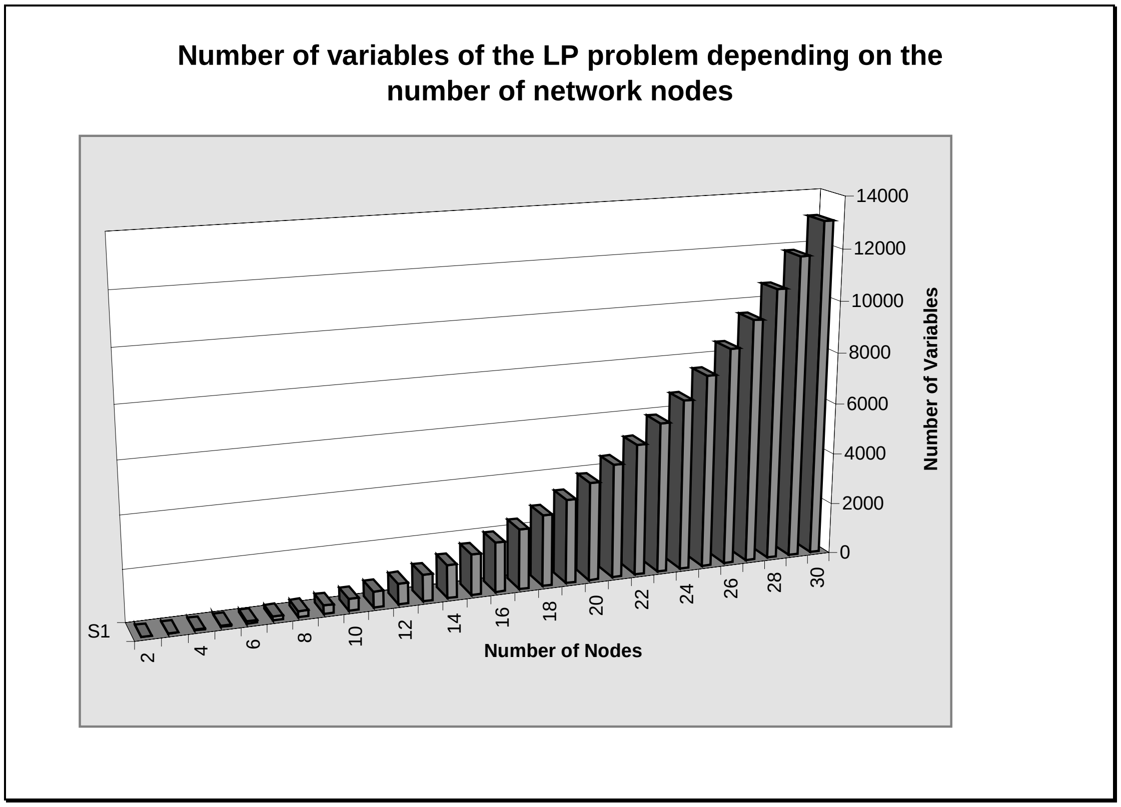


Fig. 3.3 : The number of variables of the LP problem against the number of nodes

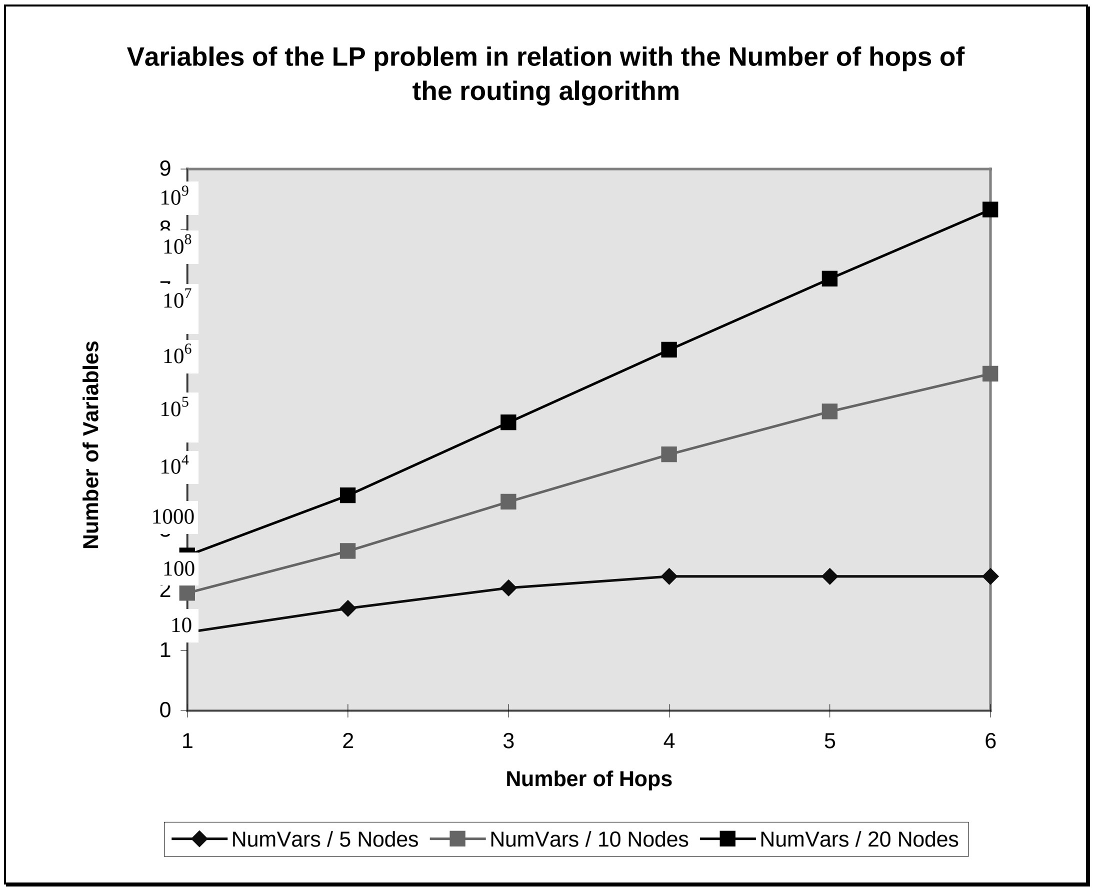


Fig. 3.4. : The number of variables of the LP problem against the number of hops used by the routing algorithm

These two figures prove that the number of variables increases dramatically with the number of nodes or hops. It should be noted that this is a *key limitation of LP*, while other optimisation methods such as heuristics, GAs etc. can tackle large problems, giving “reasonable” results. Now, the reason that no more than 20 nodes and 3-hops have been tested during the experimental part, can be clearly understood.

3.4. Implementation & Results

Let us assume that we are given the locations of the 5 switching nodes of fig. 3.1 network (x, y co-ordinates) :

	Node 0	Node 1	Node 2	Node 3	Node 4
x	18.66	14.40	17.4	13.02	18.31
y	11.54	9.88	7.37	10.65	8.96

Table 3.2 : The locations of the 5-nodes network of fig 3.1

Given the locations of the nodes, the distance between each node pair can be calculated by using the equation (3.2-8) :

Nodes	**0**	**1**	**2**	**3**	**4**
0	0	914.4	871.241	1141.96	520.726
1	914.4	0	782.307	316.057	803.355
2	871.241	782.307	0	1094.4	366.399
3	1141.96	316.057	1094.4	0	1110.68
4	520.726	803.355	366.399	1110.68	0

Table 3.3 : The Distance Matrix for the 5-nodes network of fig 3.1

An implementation the LP problem formed by objective function and the sets of constraints given in the previous paragraph (the LP problem is constructed by the code presented in *Appendix E*), into the 5-nodes network, results in a list, presented in *Appendix C*, which will constitute the input of the *lp_solve_2.0* program. It should be mentioned that 2-hop path routing has been assumed, meaning that the total number of paths (direct and 2-hop paths) in the network, according to the equations (B.2-1), (B.3-1) and (B.3-2), will be *40*.

It should be emphasised that as the variables of the LP problem has to use the “x” symbol (in lp_solve_2.0), the indices of the variables representing the paths start from the value of 10 (as the last of the links variables is equal to 9).

Giving the constraints listed above as input to *lp_solve_2.0* we get the following results (concerning the values of the variables representing the links capacities), which form the optimal solution for the problem (the values that minimise the objective function) :

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
5	6	7.5	3.5	7.2	2	5	8.5	2	5.5

Table 3.4 : The optimal solution for the LP problem constructed for the case of the 5-nodes network of fig. 3.1

As it is obvious from the table above, apart from the fact that the final network topology will be a full-mesh network (as all the links capacities have non-zero values), the capacity of each one of the links is equal to the traffic between the nodes it connects. Namely, *none of the available 2-hop paths is used*. That can be also seen by examining the values of the 2-hop paths variables (that are not listed here), which are all set to 0. The cost of the network (the value of the objective function), calculated in accordance with the equation (3.2-7), is 46,717.

In order to test the LP optimisation process effectiveness, it has been tried to implement the same set of constraints in a "real world network" : the *European Optical Network* (*EON*). An overview of EON has been presented in the introductory part of the report.

The traffic matrix of EON, together with the names and the locations of the nodes have been the input of the program that creates the objective function and the constraints of the LP problem, which, in turn, have been the input of the lp_solve_2.0. The output of the later has similar form as for the 5-nodes network. The network cost is equal to 877,226. The traffic matrix and the values of the variables representing the links of EON are presented in Appendix C.

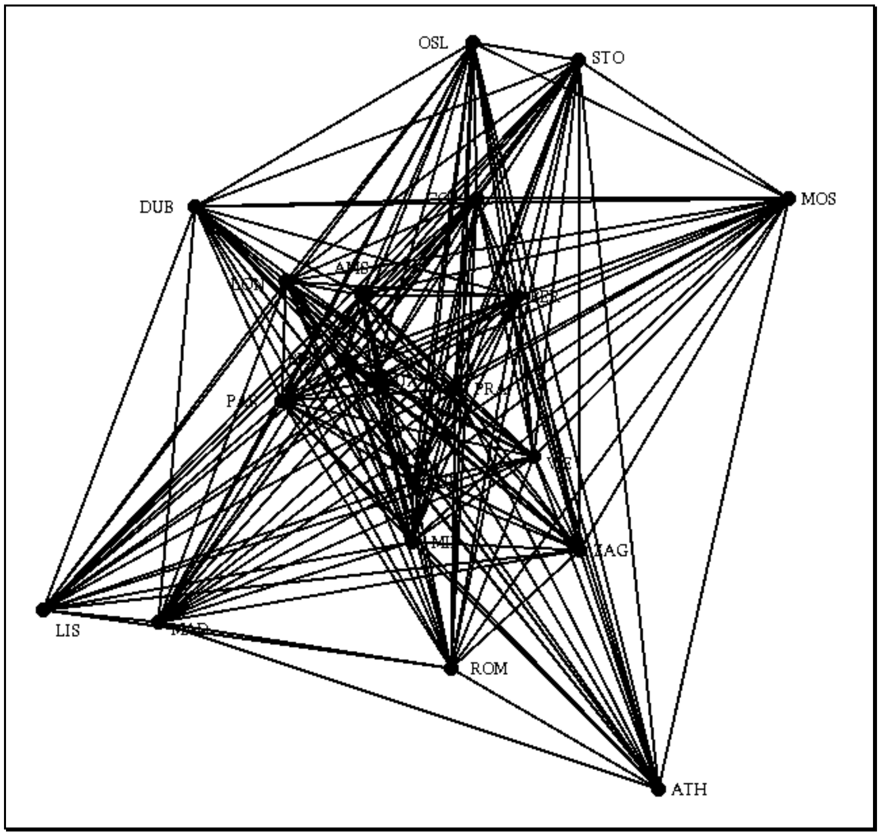


Fig. 3.5 : The resulting optimised topology of EON

It can be seen that all the network node pairs are connected with a direct link, apart from the pairs Athens-Lisbon, Dublin-Prague, Rome-Milan, Luxembourg-Prague and Lisbon-Prague, as there is not any traffic between them. Thus, according to the LP optimised solution , the topology of EON, illustrated in figure 3.5, includes *185 links* (the number of node pairs is 190).

3.5. Assessment of LP Virtual Topology Design Results - Alternative approach of the problem - Further work

Fig. 3.5 gives a clear picture of the LP failure to give a satisfactory solution to the problem that is currently examined. This is because a fully connected network would not be what someone would consider as optimum topology. The reason that leads to this problem is that there is not any way of including into the objective function the *cost of link establishment*. The results obtained are discussed in more detail in Chapter 5.

In order to include link establishment cost, an *alternative cost model*, different from that of the equation (3.2-7), would have to be implemented. That cost model would include a constant, representing the establishment cost, added to the link cost, *in case that this link would exist*. However, this approach cannot be formulated as an LP problem, as it would presuppose the use of some *decision variables*, corresponding to each one of the links and would be set equal to 1 in case that the link existed or equal to 0, otherwise. Still, that would break the *linearity of the problem*. Hence, it seems that the formulation of an IP problem would be the ideal solution to the problem. This IP approach is also presented in Chapter 5.

In case that it would be tried to add the constant value mentioned above, in equation (3.2-7), and, thus, obtaining a new objective function :

$$C_{Net} = \sum_{i=0}^{NumOf\ Links} [(2 + L_i) * x_i + C] \qquad (3.5\text{-}1)$$

where C is the cost of establishing a link, the results of the optimisation process would be identical with the previous ones (apart from the final value of the network cost which would have been increased by the constant C multiplied with the number of node pairs).

In the way that the problem has been already set, the final result (the fact that regardless of the number of hops of the routing algorithm, there are not any 2 or 3-hop paths used, so every single direct link exists in the final topology) had to be expected. This can be better explained graphically, by examining the figure 3.6, which compares two alternative topologies for a 3-nodes network, which is the simplest possible network.

In fig.3.6 (a), there is a link corresponding to each of the node pairs, whereas in fig.3.6 (b) one of the links has been removed and the traffic corresponding to the node pair A-C is routed through the 2-hop path A-B-C.

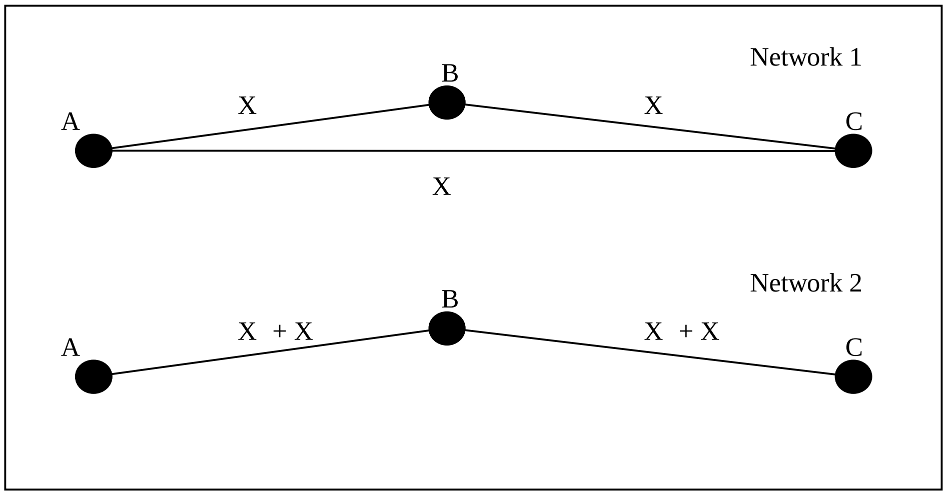


Fig. 3.6 : (a) A 3-nodes network where only the direct links are used to carry the traffic (b) The same network where only the 2 out of 3 links exist : Node pairs A-B and B-C use the direct links, whereas A-C uses a 2-hop path to carry the traffic (A-B-C)

Thus, if we assume length difference equal to ΔL, between the A-B-C and A-C paths ($\Delta L = L_{A\text{-}B\text{-}C} - L_{A\text{-}C} = L_0 + L_1 - L_2$), then, according to equation (3-2.4), the cost of the two networks will be the sum of costs of the individual links and nodes:

<u>Network 1</u> : $C_{Net1} = C_{link\ 0} + C_{link\ 1} + C_{link\ 2} + C_{Node\ A} + C_{Node\ B} + C_{Node\ C}$

$= L_0\ x_0 + L_1\ x_1 + L_2\ x_2 + (x_0 + x_2) + (x_0 + x_1) + (x_1 + x_2) =$

$= (2 + L_0)\ x_0 + (2 + L_1)\ x_1 + (2 + L_1)\ x_1$

<u>Network2</u> : $C_{Net2} = C_{link\ 0} + C_{link\ 1} + C_{Node\ A} + C_{Node\ B} + C_{Node\ C}$

$= L_0\ (x_0 + x_2) + L_1\ (x_1 + x_2) + (x_0 + x_2) + (x_0 + x_1 + 2\ x_2) + (x_1 + x_2)$

$= L_0\ x_0 + L_1\ x_1 + (L_0 + L_1)\ x_2 + (x_0 + x_2) + (x_0 + x_1) + (x_1 + x_2) + 2\ x_2$

$= L_0\ x_0 + L_1\ x_1 + L_2\ x_2 + \Delta L\ x_2 + (x_0 + x_2) + (x_0 + x_1) + (x_1 + x_2) + 2\ x_2$

$= (2 + L_0)\ x_0 + (2 + L_1)\ x_1 + (2 + L_1)\ x_1 + \underline{\Delta L\ x_2 + 2\ x_2}$

As it is clear from the above general case, the cost of the second network is greater to the cost of the first (the difference in costs is equal to the underlined factor, appearing in the second network cost).

Nevertheless, in case that the implementation of the alternative cost model, which has been referred above, is attempted, the previous conclusion will not be valid any more. Namely, if we define a link establishment cost equal to C, then the two networks cost will then be :

Network 1 : $C_{Net1} = (2 + L_0)\, x_0 + (2 + L_1)\, x_1 + (2 + L_1)\, x_1 + \underline{3\ C}$

Network 2 : $(2 + L_0)\, x_0 + (2 + L_1)\, x_1 + (2 + L_1)\, x_1 + \underline{\Delta L\ x_2 + 2\ x_2 + 2\ C}$

If, now, C was a large constant value, then *the cost of the first network would be significantly greater to the cost of the second.*

Consequently, there could be an alternative approach to this problem, in order to get by any LP limitations. Specifically, as the change of the objective function and constraints formulation is not possible, there could be added some further constraints that would force the LP solving program to use a smaller number of links, in the final network topology. These additional constraints could be set on the *node degree* (number of attached nodes). Alternatively, it could be tried to cut links which connect nodes with small traffic capacity between them, links with length much greater to the average length, or combination of those two. Of course, the resulting network topology, apart from including smaller number of links, would correspond to an objective function with a larger value, compared to its previously obtained value (= 877,226). Nevertheless, if the new objective function (equation (3.5-1)) would have been applied, and given that the constant C would have a large value, the network cost would have been significantly reduced, compared to the previous case.

This new approach could be implemented in the following way : Initially apply the constraints mentioned above, identify the links which should be cut and then run the LP solving program (lp_solve_2.0), requiring, apart from the other constraints, those links capacities *to be zero*.

An attempt of implementing this alternative approach has been made, by cutting, for each one of the nodes, all the links attached to it, with length over the average length of the attached links, multiplied with a factor, and traffic less than the average traffic of the attached links, multiplied with another factor. Various combinations of those factors have been tried. For some of these combinations the LP problem was unfeasible (no solution could be given). If, for example, the first factor is selected equal to 1.3 and the second equal to 0.8 (for every node of the network, all the links with length greater to 1.3 * average length and traffic less than 0.8 * average traffic are cut), then, according to the traffic and distance matrices there are *29 links* which will be required to have zero capacity in the final network topology.

Therefore, the resulting optimised network will include *146 links* (as, apart from the 29 links which have been cut, there are 5 node pairs with no traffic between the nodes and, thus, don't need to be connected directly) and its cost (value of the objective function) is <u>878,829</u>. The topology of that network will be the one presented in the figure following :

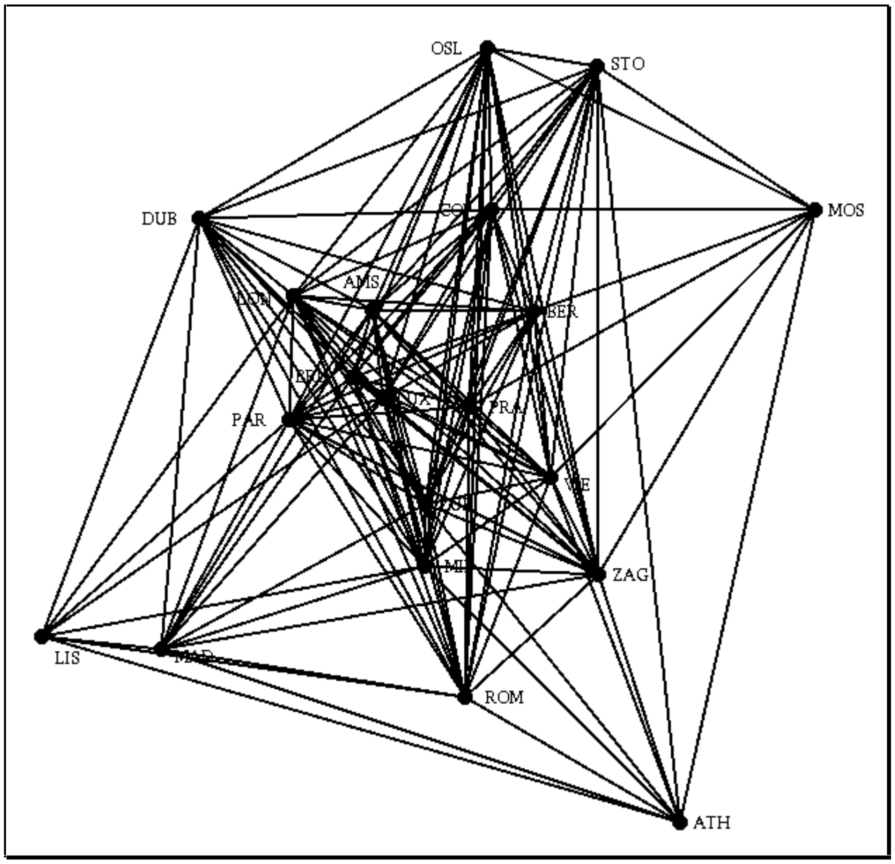


Fig. 3.7 : The resulting optimised topology of EON, when 29 links have been cut

The cost of this network is slightly greater to the value obtained without adding the new constraints (877,226). If, however, the new objective function of equation (3.5-1) is applied, and the link establishment cost is estimated to be C = 100, the new costs will be :

185 links Network : C_{185} = 877,226 + (185 * C) = 895,726

146 links Network : C_{146} = 878,829 + (146 * C) = 893,429

Obviously, the 146 links network have a considerably lower cost, compared to the mesh network, if a more realistic cost function is applied. Even better results might be obtained for different combinations of the two factors mentioned above.

Of course, the traffic corresponding to the pairs of nodes not directly connected will have to be routed through 2-hop paths. For each of those node pairs there are 18 2-hop paths that can be used. The selected path will be the one with the minimum cost, namely the minimum length. Thus, there are 40 2-hop paths arising. Those paths are presented in Appendix C. It can be seen that for each of the 40 2-hop paths appearing, the intermediate node is very closed to the line defined by the end nodes (the length of the 2-hop path is the minimum possible).

3.7. Backup Paths

A further investigation of the network design optimisation process that was implemented, would be the introduction of *backup paths*. Those paths would connect, as in the previous case, a pair of node, traversing a different series of links. Thus, each node pair in the network would be connected by two separate paths which would not share any links (none of the intermediate nodes would be in common).

Backup routes are usually established in networks, in case that safety and reliability are considered as crucial issues. Thus every time that one of the links or the intermediate nodes of the primary path would fail, the traffic between the end nodes would be routed through the backup route. Each backup route has, therefore, to be able to carry the traffic between those nodes.

As far as the LP approach of the problem is concerned, there are some new variables and some new constraints that have to be added in order to cope with the problem examined so far. In particular, each of the new variables representing the backup paths: z_i , i = 0,1, ... Num Of Paths, will correspond to the same path that the primary paths variables y_i do. The new constraints will be similar to those set for the primary paths :

♦ *Constraint on the sum of backup paths variables corresponding to a specific node pair*: As previously, for each pair of nodes, there is a variable z_i assigned to the direct, the 2, 3, ... -hop paths, depending on the number of hops used by the routing algorithm. Consequently, there is a requirement for the sum of traffic units carried by those streams to be equal to the traffic between the two nodes. This constraint produces a set of equalities, *equal to the number of node pairs* [46].

Thus, if backup paths are introduced in the 5 nodes network, and a 2-hop routing algorithm is implemented, we would have the following backup paths variables, corresponding to the Node 0 - Node 1 node pair :

– Node 0 - Node 1	:	z_0
– Node 0 - Node 2 - Node 1	:	z_1
– Node 0 - Node 3 - Node 1	:	z_2
– Node 0 - Node 4 - Node 1	:	z_3

In this example, the sum of these variables should give the traffic between nodes 0 and 1, which according to the traffic matrix (Table 3.1) will be :

$$z_0 + z_1 + z_2 + z_3 = 2 + 3 = 5$$

♦ *Constraint on the sum of paths variables sharing the same link* : As in the primary paths case, it is required that the sum of the streams sharing the same link must be equal to the link capacity. Of course, in this case, apart from the primary, backup paths are included as well. The number of equalities generated as a consequence of this constraint will be *equal to the number of links* [46].

In the 5-nodes network, for example, the link x_0 (that connects the nodes 0 and 1), can be used by the paths :

– Node 0 - Node 1	:	$y_0,\ z_0$
– Node 0 - Node 1 - Node 2	:	$y_5,\ z_5$
– Node 0 - Node 1 - Node 3	:	$y_9,\ z_9$
– Node 0 - Node 1 - Node 4	:	$y_{13},\ z_{13}$
– Node 1 - Node 0 - Node 2	:	$y_{17},\ z_{17}$
– Node 1 - Node 0 - Node 3	:	$y_{21},\ z_{21}$
– Node 1 - Node 0 - Node 4	:	$y_{25},\ z_{25}$

Thus, the sum of those variables must not exceed the capacity of the link that connects the nodes 0 and 1 (x_0) :

$$y_0 + y_5 + y_9 + y_{13} + y_{17} + y_{21} + y_{25} + z_0 + z_5 + z_9 + z_{13} + z_{17} + z_{21} + z_{25} = x_0$$

Nevertheless, the requirement of having disjoint primary and backup paths has not been met yet, with the constraints set so far. Thus, there is need for an extra constraint:

♦ *Constraint that implements the requirement for node-disjoint paths* : As there are two variables (y_i , z_i) representing a primary and a backup connection over the same path, it is required not to let both of these variables to have non-zero value. Namely, in case that two end nodes are able of routing the traffic between them through several paths, the LP solving problem should be forced to use separate paths for the primary and the backup paths. This can be achieved, if *the sum of y_i and z_i is required to be less or equal to the traffic between the end nodes* to which those variables correspond. The complexity of the problem increases in case that 3, 4 or more hops routing is applied. Then, there are different paths that might share the same link (if one of the intermediate nodes is common), so paths have to be checked link-by-link in order to decide whether they are node-disjoint or not. As a consequence of this constraint, a number of inequalities, equal to the number of paths will be produced [46].

In the 5-nodes network example, the implementation of this constraint on the Node 0 - Node 1 node pair, will give :

$y_0 + z_0 <= 5$

$y_5 + z_5 <= 5$

:

$y_{25} + z_{25} <= 5$

3.6.1 Further improvement on backup paths problem approach

In addition to the constraints described above, it is essential that the LP problem solution will include primary routes which use less hops comparing to the corresponding backup routes. That, of course, will not reduce the network cost, but it is necessary for two reasons : Firstly, as for every pair of nodes the primary route is the most commonly used to carry the traffic, it is important (for network reliability purposes) to *reduce the probability of primary route fail*. A fail might occur in case that one of the paths components (one of the links or the intermediate nodes included) fails. Hence, the less the hops, the less the number of path components and, thus, the less the probability of failing. Secondly, the less the hops of a path, the less the delay observed for the traffic carried (as the number of switching nodes and, thus, *the switching delay is reduced*).

The idea expressed above (primary paths with less number of hops than the backup paths), can be applied by *modifying the objective function* of the LP problem : apart from the network cost (the cost of nodes plus the cost of links) the cost of primary as well as the backup paths should be also included. What is of great importance, is to force the LP solution assign the paths with the greater number of hops to the backup routes. That can be achieved by multiplying with an *weighting factor* the cost of primary paths, and, thus, making them more "expensive". Consequently, since the primary paths are going to be used anyway, the solution of the LP problem will include primary paths with the least number of hops, in order to minimise the overall cost. In case that this idea is not applied, the assignment of a path as a primary or backup route will be done randomly.

Thus, the implementation of the modified objective function to the EON, for a value of that factor equal to 2 (the primary paths cost is the double of that of the backup paths), will give a solution of a 188 links network, with 184 direct and one 2-hop primary connections (the equivalent numbers for the backup routes are one direct and 184 2-hop connections). The list of the 2-hop paths selected for the backup connections is presented in *Appendix C*. As in the previous case, where no backup paths were under consideration, it can be observed that in every case the 2-hop path selected, is the one with the minimum length, among all the available 2-hop paths.

Moreover, the total primary paths capacity is equal to the equivalent of the backup paths (due to the constraints set), while the total primary paths cost is less than the total backup paths cost, where the cost is calculated according to its definition in relation with the paths capacity and length. Naturally, the network cost, given by equation (3.2-7) (= 1,920,730) is greater to that of the same network without any backup paths.

3.6.2. Alternative Approach for Backup Paths Case

The number of links existing in the final “optimum” topology (188 links) is extremely large. Thus, the same technique discussed in section 3.5 (to cut all the links that don’t meet the requirements for traffic and length being under a certain level) should be used in this case as well, in order to limit the number of links in the network.

Alternatively, the formulation of an equivalent IP problem could be attempted. This approach is presented in Chapter 5.

CHAPTER 4

ROUTING & WAVELENGTH ASSIGNMENT PROBLEM

4.1. Introduction

The second part of the experimental work deals with the *use of Linear Programming techniques for routing and wavelength assignment, in an all optical network*.

In contrast to the previous problem examined (virtual topology design), in this case the objective is not to achieve an optimum topology, as *the topology is already fixed*. Apart from the topology, the problem input also consists of the network Traffic Matrix as well as the switching nodes names and geographical locations (co-ordinates).

The network is considered to use *WDM* (Wavelength Division Multiplexing). Namely, a number of channels, each of which corresponds to a specific wavelength λ_i, is available for each of the existing network links. Of course, it is possible to have different number of channels available in different links. The *objective of the optimisation process* is to *minimise the number of channels required in the network*[5]. That could be translated into *minimising the total capacity of links*, as this is directly affected by the number of network channels.

From what have been already mentioned so far, it can be concluded that apart from the existing links capacities, in order to meet an appropriate LP problem formulation, there should be some *additional variables representing the wavelengths used for each of the links*. The construction of the LP problem will take place in the paragraphs following.

4.2. Modelling considerations

The procedure of constructing an LP problem for wavelength assignment, can be followed and tested with more ease by trying to apply it onto a "real" network. This could be a limited version of European Optical Network, which includes only 9 central nodes : *Paris*, *London*, *Berlin*, *Milan*, *Brussels*, *Amsterdam*, *Prague*, *Zurich* and *Luxembourg*.

[5] An alternative could be to reduce the maximum number of wavelengths per fibre. A smaller number of different wavelengths in a fibre would be desirable from a physical optics viewpoint.

Only 17 out of the maximum attainable number of links (= 9 * (9-1) / 2 = 36 links) exist in the given topology. Those links are shown in fig. 4.1.

In order to model the problem in a way that will be much easier to handle, a new structure, called Adjacency Matrix, is introduced. This is an m x m array, with empty forward diagonal, which indicates whether a connection between a pair of nodes is established, or not. Hence, according to the connections listed above, the Adjacency Matrix for the 9 central nodes of EON will be :

Nodes	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
0	-	1		1	1			1		**Paris**
1	1	-			1	1				**London**
2			-		1	1	1	1		**Berlin**
3	1			-			1	1	1	**Milan**
4	1	1	1		-	1			1	**Brussels**
5		1	1		1	-			1	**Amsterdam**
6			1	1			-	1		**Prague**
7	1		1	1			1	-		**Zurich**
8				1	1	1			-	**Luxembourg**

Table 4.1 : The Adjacency Matrix for the 9 central nodes of EON

Thus, the network topology will be the one illustrated in figure 4.1 :

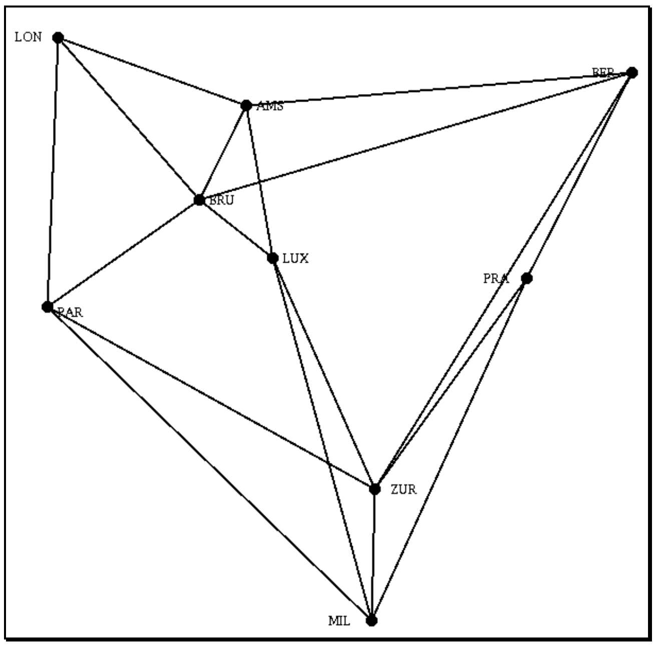


Fig. 4.1 : Topology of the 9 central nodes of EON

It is assumed that *only 2.5 Gbps channels are available*. In addition, it should be mentioned that for each pair of nodes only the *4- shortest paths* are considered, so that the complexity of the problem will be reduced. These 4 shortest paths are found in terms of distance, not minimum number of hops. Each of the paths in the network *uses a single wavelength to carry information end-to-end*. Thus, no wavelength conversion is needed by the switching nodes.

The algorithm used in order to trace the 4-shortest paths is rather simple : Firstly, for each of the 36 pairs of nodes, all the existing direct, 2,3 and 4-hop paths are found, then they are sorted in increasing order of length, and, finally, the 4 paths with the minimum distance are selected. The whole list of the 4 shortest paths for each of the node pairs on the network is presented in *Appendix D*. By limiting the considered number of paths per node pair, the number of variables of the LP problem as well as the problem size are dramatically reduced. Apart from that, in any case, the number of wavelengths used in the network will be increased, if longer paths are used [5].

The traffic matrix is given in the table following :

Nodes	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
0	-	22.45	23.3	8.6	17.2	6.52	0.42	16.69	1.96	**Paris**
1	19.06	-	19.06	4.58	5.85	9.74	0.51	6.27	1.05	**London**
2	26.27	25.42	-	11.61	9.32	19.74	4.74	24.91	2.29	**Berlin**
3	8.0	3.82	8.81	-	1.7	1.23	0.38	5.76	0.21	**Milan**
4	17.2	6.44	9.32	1.82	-	14.32	0.17	1.86	2.54	**Brussels**
5	6.52	11.52	20.17	1.4	14.57	-	0.34	2.46	0.51	**Amsterdam**
6	0.34	0.34	2.63	0.26	0.17	0.25	-	0.42	0	**Prague**
7	20.42	6.69	28.72	9.15	2.2	2.97	0.85	-	0.51	**Zurich**
8	2.71	0.85	2.8	0.34	3.22	0.51	0	0.42	-	**Luxembourg**

Table 4.2 : The Traffic Matrix for the 9 central nodes of EON

Given the locations of the nodes, a Distance Matrix can be also constructed, as it has been done with the simple, 5-nodes network, in the previous chapter.

4.3. Formulation of the LP problem

4.3.1. Objective Function

As it has been already discussed, the minimisation of the network channels and the minimisation of the total links capacity could be considered as equivalent problems. Thus, for the specific problem of 9 central nodes of EON, the form of the objective function would be :

```
min:  x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9 + x10 + x11 + x12 + x13 + x14 +
x15 + x16;
```

where x_0 - x_{16} represent the capacities of the 17 established links in the network.

4.3.2. Link constraints

As it was the case with the previous problem, that of virtual topology design, there are some constraints that should be set, concerning the links of the network. The only difference is that in this case the initial topology is not a fully-meshed topology, and, thus, only the existing links should be considered. Specifically :

♦ The group of inequalities concerning the *constraint on "cuts around nodes"* are presented below. The number of inequalities will be equal to the number of network nodes (= 9).

```
Node 0 :    x0 + x1 + x2 + x3 >= 192.99;
Node 1 :    x0 + x4 + x5 >= 127.94;
:
Node 8 :    x12 + x14 + x15 >= 18.99;
```

♦ Similarly, the number of inequalities concerning the c*onstraint on "cuts around links"* will be equal the number of existing links (= 17) :

```
Link 0 :    x1 + x2 + x3 + x4 + x5 >= 258.29;
Link 1 :    x0 + x2 + x3 + x10 + x11 + x12 >= 232.13;
:
Link 16 :   x3 + x8 + x9 + x10 + x11 >= 139.58;
```

4.3.3. Path Constraints

Constraints concerning paths are, again, treated in the same way with the virtual topology design case, except of the fact that there are not any variables representing the paths any more, but only a number of variables corresponding to all the wavelengths traversing a single path. Namely, for all the 4 shortest paths connecting a pair of nodes, there is a fixed a number of variables N, representing the channels (wavelengths) $\lambda_{0,}$ λ_1, ... , λ_{N} , which are available for transferring the traffic between the two end nodes. In case that this traffic exceeds the threshold of the 2.5 Gbps (channel capacity), then more than one channels will be utilised.

Thus, the same constraints which were valid for the variables corresponding to paths, will, now, be applied on the variables representing wavelengths :

♦ *<u>Constraint on the sum of channels variables corresponding to a specific node pair</u>* : The sum of the variables values of corresponding channels between two end nodes, should be equal to the traffic between those nodes [46]. There will be a number, equal to the number of node pairs, of equalities posed by this constraint. Thus, in case that 4 distinct wavelengths (λ_0 , λ_1 , λ_2 and λ_3) are available in the network, then, there will be 16 variables (4 wavelengths per each of the 4 shortest paths) corresponding to each of the node pairs. The implementation of this constraint onto the node pair Paris - London will give :

```
x17 + x18 + x19 + x20 + x21 + x22 + x23 + x24 + x25 + x26 + x27 + x28 + x29 + x30 +
x31 + x32 = 41.51;
```

The variables x_{17}-x_{20} are assigned to the path *Paris - London*, the x_{21}-x_{24} to the path *Paris - Brussels - London*, the x_{25}-x_{28} to the path *Paris - Brussels - Amsterdam - London* and the x_{29}-x_{32} to the path *Paris - Brussels - Luxembourg - Amsterdam - London*. These paths are the 4 shortest available paths for the node pair Paris - London. The whole list of the 4 shortest paths for each node pair (of the 9 central nodes network version of EON), their length and the channel variables corresponding to these paths is presented in Appendix D. It should be mentioned, that, as previously, the indices of the variables representing the channels start from 17 (as the last link variable has index equal to 16).

♦ *<u>Constraint on the sum of channel variables sharing the same link</u>* : All the channel variables which traverse the same link should sum up to the capacity of the link. A number of equalities, equal to the number of existing links, will be posed as a consequence of this constraint [46].

For example the capacity of the link *Berlin - Amsterdam* (x_7) will be affected by the utilisation of the channels corresponding to the paths *Paris - Brussels - Amsterdam - Berlin* (variables x_{37}-x_{40}) , *London - Amsterdam - Berlin* (x_{145}-x_{148}), *London - Brussels - Amsterdam - Berlin* (x_{153}-x_{156}), ..., *Prague - Berlin - Amsterdam - Luxembourg* (x_{573}-x_{576}). Hence :

```
x37 + x38 + x39 + x40 + x145 + x146 + x147 + x148 + x153 + x154 + x155 + x156 +
......... + x573 + x574 + x575 + x576 = x7;
```

4.3.4. Wavelength Constraints

Apart from the constraints set above, which have many similarities with the virtual topology design problem, there are some other that should be added, in order to introduce the limitations relative to the nature of an optical network, into the LP problem. Those constraints will be :

♦ *Constraint on the channel capacity* : All the variables representing the channels capacities should be kept below the limit of the 2.5 Gbps. This is translated into setting a number of inequalities, equal to the total available number of channels in the network [46]. Thus, for the 4 distinct wavelengths network case this constraint will give :

```
x17 <= 2.5;
x18 <= 2.5;
:
x592 <= 2.5;
```

♦ *Constraint which aims to avoid any potential wavelength blocking* : Wavelength blocking occurs when two or more distinct paths, which have a specific link in common, use the same wavelength (i.e. λ_1) in order to forward the traffic of the node pairs they correspond to. This case is depicted in figure 4.2. In this case, it should be required that only one of the variables representing a single wavelength over a specific link can have a non-negative value (and less than 2.5 Gbps). This, in terms of a restrictive LP model, is translated into keeping the sum of these variables under the limit of 2.5 Gbps. The number of inequalities posed as a consequence of this constraint is equal to the number of the existing links multiplied with the number of wavelengths available in the network [46].

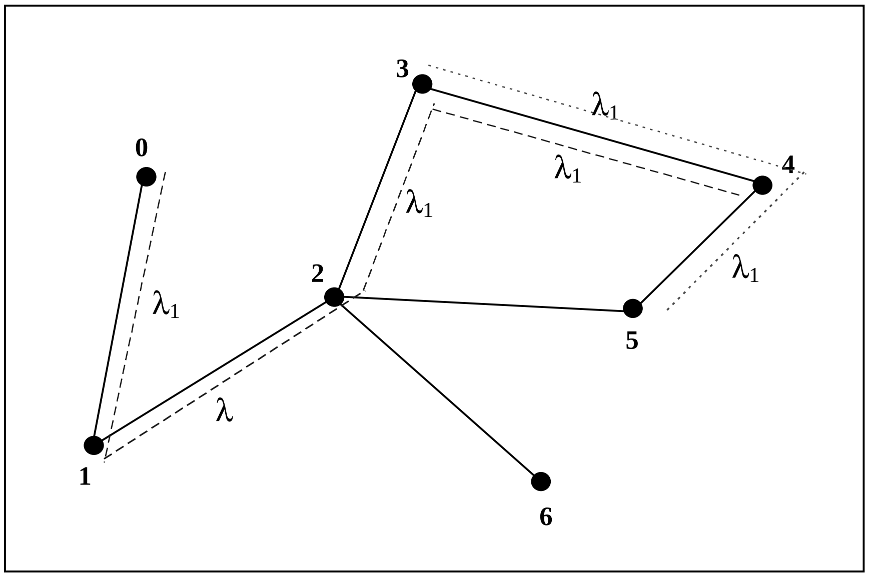


Fig. 4.2 : Blocking case : The connections 0-1-2-3-4 and 3-4-5 use the same wavelength λ_1 (overlapping across the link 3-4)

The implementation of this constraint onto the link *Berlin - Amsterdam* (x_7) will give :

λ_0 :	x37 + x145 + x153 + x209 + x277 + x281 + x285 + x289 + x341 + x345 + x457 + x497 + x573 <= 2.5;
λ_1 :	x38 + x146 + x154 + x210 + x278 + x282 + x286 + x290 + x342 + x346 + x458 + x498 + x574 <= 2.5;
λ_2 :	x39 + x147 + x155 + x211 + x279 + x283 + x287 + x291 + x343 + x347 + x459 + x499 + x575 <= 2.5;
λ_3 :	x40 + x148 + x156 + x212 + x280 + x284 + x288 + x292 + x344 + x348 + x460 + x500 + x576 <= 2.5;

4.4. Results

The LP problem, the formulation of which has been described in the previous paragraph, has been constructed by the program presented in *Appendix E*, and given as input in lp_solve_2.0. In particular, this program takes as input a number of wavelengths, in order to formulate the LP problem with the right number of variables. This is the number of channels available for each one of the links. However, it is not necessary that all of the channels will be used. Thus, several numbers of wavelengths have been given as input.

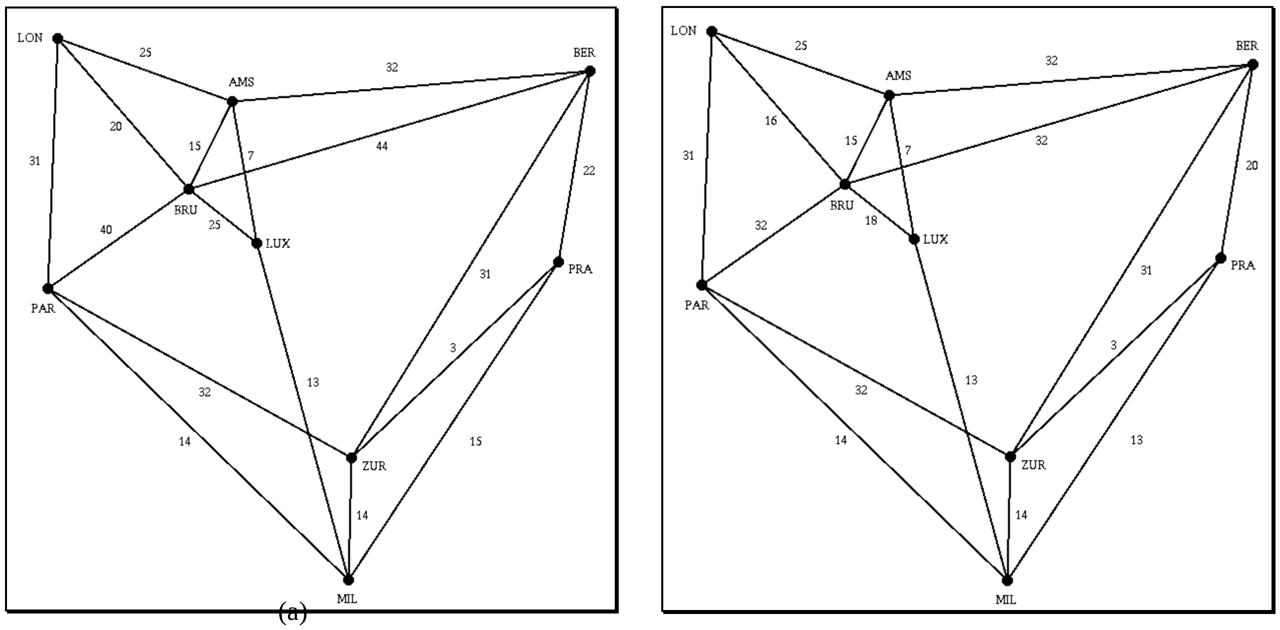
The smallest number that gave a LP problem with feasible solution was *32 wavelengths*. In that case, a total of 36 * 4 * 32 = 4608 channels were available in the network (the first term of the product represents the number of node pairs, the second the number of paths per node pair and the third the number of wavelengths per path).

Nevertheless, the output of lp_solve revealed that the wavelength blocking constraint (see section 4.3.4) is not really satisfied, as it should have been expected. Namely, although the sum of the variables representing a single wavelength over a specific link is kept below the limit of the 2.5 Gbps, there are cases where two or more of these variables have non-negative values. However, it seems that this cannot be avoided, as the linearity constraint of LP would not allow to require *only one* of the variables to have non-negative value. Hence, the formulation of an equivalent IP problem would give again an ideal solution. An IP approach for this problem is presented in Chapter 5.

Concerning the lp_solve output, out of these maximum available number of 4608 channels only *348* have been used. Still, there is a number of channels (wavelengths) used to carry information associated with two or more distinct node pairs. Hence, the total number of wavelength variables with non-negative value is equal to *383*. For example, there are 40 paths active over the link Paris - Brussels, using only the 32 available wavelengths. The number of paths corresponding to each of the wavelengths is:

λ_0:2, λ_1:1, λ_2:1, λ_3:2, λ_4:1, λ_5:1, λ_6:1, λ_7:1, λ_8:1, λ_9:2, λ_{10}:1, λ_{11}:1, λ_{12}:1, λ_{13}:2, λ_{14}:1, λ_{15}:1, λ_{16}:1, λ_{17}:1, λ_{18}:2, λ_{19}:1, λ_{20}:1, λ_{21}:2, λ_{22}:1, λ_{23}:2, λ_{24}:1, λ_{25}:1, λ_{26}:1, λ_{27}:2, λ_{28}:1, λ_{29}:1, λ_{30}:1, λ_{31}:1

The way that these channels are distributed to each one of the established links, is illustrated in fig. 4.3 (a). However, assuming that a *combination of WDM/OTDM* (details for the combination of these two multiplexing techniques have been mentioned in section 2.4) is implemented in the network, then it is possible for two or more source-destination (s-d) pairs to use the same channel (wavelength) in order to transmit data over the same link (using time division technique). In that case, the limited number of *348 channels* can be used to satisfy the traffic requirements for the *383 s-d pairs*. The distribution of channel corresponding to that case is presented in fig.4.3 (b).



(b)

Fig.4.3 : Distribution of channels in the 9 central nodes of EON (a) Conventional approach (b) Use of WDM/OTDM

The total number of channels used in the network can be further decreased using a more sophisticated technique described in *Appendix D* (section D.2). With this technique, a more efficient use of the existing channels is achieved, so there is need for only *335 channels*. This is a significant progress, since the total number of optical transmitters and detectors needed in the network will be also reduced (it is known that in WDM networks there is need for a laser - optical detector pair for each wavelength used in an optical path). This will have a serious effect in network cost, which will be reduced as well.

4.5. Further Work - Non Wavelength Blocking Solution

As it has been already clear by the preceding discussion, the major problem faced so far (imposed due to LP limitations) is that of wavelength blocking. Thus, a special technique had to be found in order to achieve a solution :

Initially, the *elements of traffic matrix* have been *rounded to the nearest multiple of 2.5*. Then, the variables representing the channel (wavelength) capacities have been also included in the objective function (in addition to the variables representing links capacities), assigning them different *weighting factors* (in order to force the LP solver to fill the variables values up to 2.5).

Specifically, by multiplying each variable of the objective function with different factor, we force the LP solver to assign larger values to the variables that are multiplied with smaller factors, in order to obtain the minimum possible final value for the objective function. Of course, the value of the wavelength capacity variables won't exceed the limit of the 2.5 Gbps, as this will be prevented by the wavelength constraint (described in section 4.3.4).

In case that the factors are not used, then, if for example the traffic between a source-destination pair corresponds to the capacity of N channels (= N * 2.5 Gbps), this will not necessarily use only N channels (it might be broken to more than N channels as that would not have any impact to the value of the objective function).

Finally, lp_solve_2.0 has been run for various numbers of wavelengths, and the smallest number that gives a solution where no wavelength blocking occurs is 40 (for each one of the paths, there are 40 wavelengths available). Of course, out of the overall 36 x 4 x 40 = 5760 available channels in the network, *only 347 of them are used* (each one of 347 nonnegative variables have value equal to 2.5). The distribution of channels to each one of the existing network links is presented in fig. 4.4 :

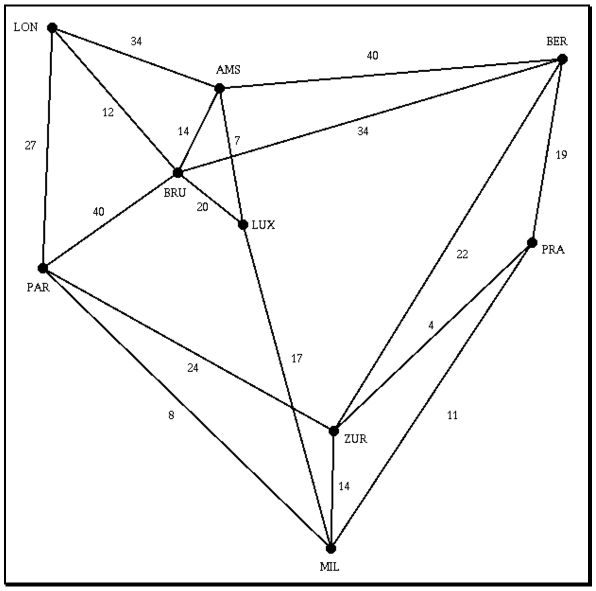


Fig.4.4 : Distribution of channels in the 9 central nodes of EON in the case that <u>*no*</u> *wavelength blocking occurs*

CHAPTER 5

DISCUSSION OF RESULTS, LIMITATIONS OF LP, IP APPROACH

5.1. Discussion of Results

It must be apparent by the preceding discussion (in Chapters 3 & 4) that Linear Programming cannot be considered as an ideal approach for network design optimisation problems.

In particular, as far as the first problem examined (“Virtual Topology Design” problem) is concerned, this can be made clear by examining fig. 3.5, in which a complete mesh topology is suggested as the “optimum” one for EON. However, this is not what common sense would consider as optimum. This is because a fully connected network is unlikely to be economically viable unless the majority of routes are adequately loaded, a condition that is more difficult to achieve as network size increases [34].

The same would be the case in the second problem (“Routing & Wavelength Assignment” problem), unless a special technique (using weighting factors) had been applied. Specifically, LP could not, initially, give a wavelength assignment solution where no wavelength blocking would occur.

Hence, in both cases, the results cannot be considered as satisfactory.

5.2. Limitations of LP

The *key limitation of LP* is that the objective function as well as the constraints included in LP problems have to be in a *strictly linear form*. This means that there cannot be any integral restrictions on some or all the variables : *every <u>real value</u> that satisfies the constraints set, is a valid one*.

Thus, in the virtual topology design problem, there is no way of including *links esablishment cost* in the objective function, as that cost would be assigned *only to the existing links* in the final optimum topology. That, in turn, would require some *decision variables* to be included in objective function, the final value of which (0 or 1) would decide whether a specific link would exist or not. However, an approach of this kind would *break the linearity of the problem*.

Other factors that possibly contribute to the fact that no satisfactory solution can be obtained, are the likely non-linear nature of link and node costs. Namely, nodes cost probably increase more than linearly with respect to the number of links (or capacity) incident, as the switching procedure becomes more complex, whereas links cost probably goes up less than linearly (economy of scale) with capacity [46].

Similarly, in the routing & wavelength assignment case, LP proves to be incapable of coping efficiently with wavelength blocking problem. This is because wavelengths capacity variables cannot be required to be either zero or equal to an integral value (maximum wavelength capacity).

Thus, it seems that Integer Programming would be the method that would provide *really optimum solutions*, with a *similar problem formulation*, although it would need much greater time period in order to complete the optimisation process. This is why IP is so widely used in the literature to cope with complex network design optimisation problems [16, 17, 18, 25, 26].

5.2. IP Approach to Virtual Topology Design Problem

As it has been already apparent, LP cannot give any satisfactory solution to the problem as it would be impossible to make the LP solving program to cut a large number of links and, thus, provide a "logical", in sight, optimum topology. What would ensure a solution would be some additional *decision variables* (their number would be equal to the number of potential links), each of which would take a value equal to 1 only if the corresponding link would exist and a value equal to 0 elsewhere.

The only optimisation method that could provide a way of applying this idea would be IP and more specifically, zero-one IP. The formulation of an IP problem is similar to an equivalent LP problem, apart from the fact that the linearity constraint is removed. The IP formulation of the problem currently under consideration would have as objective function the following :

$$\min : C_{Net} = \sum_{i=0}^{NumOf\ Links} [(2 + L_i) * x_i + C] * d_i \qquad (5.2\text{-}1)$$

where $\underline{d_i = 0 \text{ or } 1}$, i = 0, ... , Num Of Links, are the decision variables.

It is clear that the IP solving program would take into consideration the cost of link establishment C, apart from its length and capacity, and that would force it to use the minimum possible number of links that would still satisfy the constraints. It should be noticed that the LP problem constraints remain identical in the IP approach. However, the major disadvantage of IP is that it is much more time consuming comparing not only with LP, but with all the other formal methods of optimisation (Heuristics, Genetic Algorithms, Simulating Annealing, etc.). Thus, apart from the fact that use of IP was not a part of this project, it would be almost impossible, considering the time limited period available for the project work to implement IP techniques, in order to have satisfactory results.

5.3. IP Approach to Backup Paths Problem

IP could have been applied in case that backup paths were considered in virtual topology design, as well. In that case, y_i's and z_i's would be multiplied with some decision variables, d_{yi} and d_{zi} respectively, which would have as only valid values 0 or 1 and requiring :

$$d_{yi} + d_{zi} = 1, \text{ for } i = 0,1, \dots , \text{Num of Paths}$$

That way, the primary and the backup paths would have been assigned distinct routes. Then, the cost function given in (5.2-1) would have been applied, forcing the LP solver to limit the number of network links used.

5.4. IP Approach to the Routing & Wavelength Assignment Problem

It is clear that, as in the virtual topology design case, the LP method is incapable of coping with all this problem requirements, as well. The initial results cannot be considered as satisfactory, as the constraint against wavelength blocking is not really satisfied, as mentioned in previous sections. A possible constraint that would force the variables representing the wavelengths capacity to take only values equal to 0 or 2.5 would provide a solution but it would, simultaneously, *break the linearity* of the problem. Thus, integer programming would be the only method that would provide a decent solution.

The first step, in the procedure of formulating a zero-one IP problem, would be to *round all the elements of the traffic matrix to the nearest multiple of 2.5*, so that the traffic between every node pair would correspond to an integer number of 2.5 Gbps channels. Then, the new traffic matrix contents would be divided by 2.5 in order to obtain a *Channel Matrix*.

The IP problem formulation would be then, identical to that of the equivalent LP problem, with the only difference that instead of using the traffic matrix contents for putting upper or lower limits in the various constraints, now the channel matrix would be used. Likewise, the last two constraints (wavelength constraints) would have an upper limit equal to 1 (instead of 2.5). Finally, there would be an additional constraint, requiring the variables representing the channels capacities to be equal to 0 or 1 :

$$x_i = 0 \text{ or } 1, \quad i = 17, \dots , 592$$

Of course, when the optimisation procedure would finish, all the values corresponding to link variables would have to be multiplied with 2.5 in order to obtain the capacity of the links in the network with the optimal wavelength assignment.

An alternative IP approach that would provide equally satisfactory results, would be to include the variables representing the channel capacities into the objective function; then multiply these variables with a decision variable (taking values equal to 0 or 1), divide the channel variables into groups that correspond to single wavelengths over a common link, and, finally, require the sum of the decision variables associated with each one of these groups to be equal to unity. That way only one of those decision variables would be equal to 1 and the rest would be equal to 0 meaning that only one of the paths sharing the common link would be allowed to use this specific wavelength, in order to transfer information.

CHAPTER 6

CONCLUSIONS - DIRECTIONS FOR FUTURE WORK

A brief survey of network design and optimisation methods applied to it has been presented in this report. A major part of the literature survey focused on illustrating the applications of LP/IP in network design.

Concerning the experimental part of the project, LP has been applied in two cases of network design : Virtual Topology and Wavelength Assignment. The two problems were modelled for several different cases taking into consideration numerous factors, parameters and constraints. The capabilities as well as the limitations of the implementation of LP in network design problems have been investigated in depth. Results have been taken for both problems.

Although the results obtained are not what someone would consider as "optimum" (especially for the virtual topology design problem), this is due to the *weakness of this method to cope efficiently with these kind of problems*. Nevertheless, some techniques were applied in order to improve the solution given. Indeed, it has been finally managed to obtain a feasible optimum solution for the routing & wavelength assignment problem by a using a certain technique.

These are described in detail in Chapters 3 & 4, where apart from the presentation of the results, the reasons of LP failure to give an optimum "by sight" solution have been considered and IP is suggested as more successful and efficient method to deal with this kind of optimisation problems. There is also a clear indication of how IP can be used in order to give a satisfactory solution, as the equivalent IP problems have been formulated.

By what have been stated so far, it is apparent that IP could be an ideal alternative to LP method approach that would provide, in any case, optimal solutions for network design problems. However, because of the significant amount of time that IP problems require to be solved, they are not proposed to be implemented for problems of large scale (i.e. for large number of nodes or large number of hops used by the routing algorithm, in the case examined), especially if the time limitations of an MSc project are taken into account.

On the other hand, IP method can serve us a comparison with other optimisation methods, e.g. Heuristics or Genetic Algorithms, which give near optimal solutions in a relatively short amount of time. Specifically, IP can provide upper and lower bounds to the problem solution and, thus, be a measure of the other methods efficiency and reliability.

Apart from IP, it would be interesting to apply some other methods of network design optimisation, such as Heuristics or Simulating Annealing, to deal with the same two problems, as this work has been already carried out by using GAs [3,5].

Moreover, some other aspects of network design, such as maximisation of the flow on the links or minimisation of average time delay could be investigated either with LP or IP approach.

ACKNOWLEDGMENTS

The author would like to express his gratitude to Mr. M.C. Sinclair as academic and project supervisor at the University of Essex, Colchester, U.K., for his advice, support and encouragement. Many of his ideas and proposals were extremely useful and some of them have been implemented during the project work and presented in this report.

A. Proestaki, Research Officer in Dept. Of Electronics Systems Engineering, at University of Essex, is also acknowledged for providing a major part of the literature material included in the report references.

Finally, the author wishes to acknowledge Mr. Michel Berkelaar, who developed the lp_solve_2.0 program, that have been used during the experimental procedure of the project work.

The work in this project has been submitted as a partial fulfillment of the conditions for the award of a Masters of Science degree in Telecommunications and Information Systems, at the University of Essex.

REFERENCES

[1] Sinclair M.C., “Introduction to Genetic Algorithms”, Engineering and Natural Algorithms Short Course Notes, MSc in Telecommunications and Information Systems, Dept. of ESE, University of Essex, 1997.

[2] Sinclair M.C., “Genetic Algorithms for Network Optimisation”, Engineering and Natural Algorithms Short Course Notes, MSc in Telecommunications and Information Systems, Dept. of ESE, University of Essex, 1997.

[3] Sinclair, M.C. "Minimum Cost Topology Optimisation of the COST 239 European Optical Network" Proc. Intl. Conf. on Artificial Neural Networks and Genetic Algorithms, Ales, France, April 1995, pp.26-29.

[4] Sinclair, M.C., O'Mahony, M.J. "COST239: Initial Network Design and Analysis", Proc. 36th RACE Concertation, Brussels, July 93, pp.89-94.

[5] Tan, L.G., Sinclair, M.C. "Wavelength Assignment between the Central Nodes of the COST 239 European Optical Network", Proc. 11th UK Performance Engineering Workshop, Liverpool, September 1995, pp. 235-247.

[6] O'Mahony, M.J., Sinclair, M.C. & Mikac, B. "Ultra-High Capacity Optical Transmission Network: European Research Project COST 239", Information, Telecommunications, Automata Journal v12, n1-3, 1993, pp. 33-45.

[7] Griffith, P.S., Proestaki, A. & Sinclair, M.C. "Heuristic Topological Design of Low-cost Optical Telecommunication Networks", Proc. 12th UK Performance Engineering Workshop, Edinburgh, September 1996, pp.129-140.

[8] Proestaki, A. & Sinclair, M.C. "Initial Survey of Heuristics for Optical Core Network Design in ACTS WOTAN", Proc. 13th UK Teletraffic Symposium, Glasgow, Mar 1996, pp.20/1-20/8.

[9] Hewitt J., Soper A., McKenzie S., “CHARLEY : A Genetic Algorithm for the Design of Mesh Networks”, GALESIA ’95, University of Sheffield, pp.118-122, 1995.

[10] O'Mahony M. J., "Optical Communications", Network Concepts Core Course Notes, MSc in Telecommunications and Information Systems, Dept. of ESE, University of Essex, 1997.

[11] O'Mahony M. J., "Transmission Techniques", Photonic Technologies Short Course Notes, MSc in Telecommunications and Information Systems, Dept. of ESE, University of Essex, 1997.

[12] Banerjee S., Mukherjee B. Sarkar D., "Heuristic Algorithms for Constructing Optimised Structures of Linear Multihop Lightwave Networks", IEEE Transactions on Communications, Vol. 42, pp. 1811-1826, Jul 1994.

[13] Kershenbaum A., Kermani P., Grover G.A., "MENTOR : An Algorithm for Mesh Network Topological Optimisation and Routing", IEEE Transactions on Communications, Vol. 39, pp. 503-513, 1991.

[14] Yang R.J., Chuang C.H., "Optimal Topology Design Using Linear Programming", Computers & Structures, 1994, Vol. 52, pp. 265-275, 1994.

[15] Levi V.A., Calovic M.S., "Linear Programming-Based Decomposition Method for Optimal Planning of Transmission Network Investments", IEE Proceedings in Communications, Vol. 140, pp. 516-522, Nov 1993.

[16] Lowe E.D., Shepherd B., Walker N.G., "Routing and configuration of static path optical transport networks", Electronics Letters, pp. 1913-1914, Aug. 1996.

[17] Ramaswami R., Sivarajan K., "Routing and wavelength assignment in all-optical networks", IEEE/ACM Trans. Netw., Vol.3, pp. 489-500, 1995.

[18] Dutta A,. "Capacity Planning of Private Networks Using DCS Under Multibusy-Hour Traffic", IEEE Transactions on Communications, Vol. 42, pp. 2371-2374, Jul 1994.

[19] Rose C., "Low Mean Internodal Distance Network Topologies and Simulated Annealing", IEEE Transactions on Communications, Vol. 40, pp. 1319-1326, Aug 1992.

[20] Ersoy C., Panwar S., “Topological Design of Multihop Lightwave Networks”, IEEE Global Telecommunications Conference, Vol. 42, pp. 1803-1807, Nov 29- Dec 2, 1993.

[21] Wauters N., Demeester P., “Design of the Optical Path Layer in Multiwavelength Cross-Connected Networks” IEEE Journal on Selected Areas in Communications ,Vol. 14, Jun 1996.

[22] Ganz A., Gong W., Wang X., “Wavelength Assignment in Multihop Lightwave Networks”, IEEE Transactions on Communications, Vol. 42, pp. 2460-2469, Jul 1994.

[23] Hamazumi Y., Nagatsu N., Sato K., “Number of Wavelengths Required for Optical Networks with Failure Restoration”, OFC’94, San Jose, pp. 67-68, Feb 20-25, 1994.

[24] Lowe E.D., O’Mahony M.J., “Wavelength Contention Blocking in Circuit Switched WDM Optical Networks”, 20th European Conference for Optical Communications, ECOC’94, pp. 889-892, Firenze, Italy, Sept 25-29, 1994.

[25] Wauters N., Demeester P., “Wavelength Routing Algorithms for Transparent Optical Networks, 21st European Conference for Optical Communications, ECOC’95, pp. 855-858, Brussels, Belgium, 1995.

[26] Banerjee D., Mukherjee B., “A Practical Approach for Routing and Wavelength Assignment in Large Wavelength - Routed Optical Networks”, IEEE Journal on Selected Areas in Communications, Vol. 14, pp. 903-908, Jun 1994

[27] Wuttisittikulkij L., O’Mahony M.J., “A Simple Algorithm for Wavelength Assignment in Optical Networks”, 21st European Conference for Optical Communications, ECOC’95, pp. 859-862, Brussels, Belgium, 1995.

[28] Gerla M., Kleinrock L., “On the Topological Design of Distributed Computer Networks”, IEEE Transactions on Communications, Vol. COM-25, pp. 48-60, Jun 1977.

[29] Chlamtac I., Farago A., Zhang T., “Lightpath (Wavelength) Routing in Large WDM Networks”, IEEE Journal on Selected Areas in Communications, Vol.14, pp.909-913, Jun 1996.

[30] Marioka T., Kawanishi S., Takara H. Kamatani O., Yamada M., Kanamori T., Uchiyama K., Saruwatari M., “100 Gbps x 4 ch, 100 km repeaterless TDM-WDM transmission using a single supercontinentum source”, Proceedings ECOC’95, Postdeadline Paper Th.A.3.4, Brussels, 1995.

[31] Kershenbaum A., “Telecommunications Networks Design Algorithms, McGraw-Hill, Inc., 1993.

[32] Schwartz M., “Computer Communication Network Design and Analysis” Prentice-Hall, Inc., 1977.

[33] Spragins J, Hammond J., Pawlikowski K., “Telecommunications : Protocols and Design”, Addison-Wesley Publishing Company, Inc., 1991.

[34] Farr R. E., “Telecommunications Traffic, Tariffs and Costs : An Introduction for Managers”, Peter Peregrinus Ltd, 1988.

[35] Sedgewick, R., “Algorithms in C”, Addison-Wesley, 1990.

[36] William H., “Numerical Recipes in C : the art of scientific computing” (2nd ed.), Cambridge University Press, 1992.

[37] Kolman B., Beck R., “Elementary Linear Programming with Applications” (2nd ed.), Academic Press, Inc., 1995.

[38] Glover F., Klingman D., Philips N., “Network Models in Optimisation and their Applications in Practice”, John Willey & Sons, Inc., 1992.

[39] Bertsekas D., “Linear Network Optimisation : Algorithms and Codes”, MIT Press, 1991.

[40] Garvin W., “Introduction to Linear Programming” McGraw-Hill, Inc., 1960.

[41] Hu T., C., “Integer Programming and Network Flows”, Addison-Wesley Publishing Company, Inc., 1969.

[42] Schrijver, A., “Theory of Linear and Integer Programming”, Wiley, 1986.

[43] Senior, J., “Optical Fiber Communications : Principles and Practice”, Prentice-Hall, 1992.

[44] “List of Interesting Optimization Codes in Public Domain”, Internet Address : “http://ucsu.colorado.edu/~xu/software.html#lp_solve”

[45] "Action COST 239 : Ultra-high Capacity Optical Transmission Networks", Internet Address : "http://www.itr.ch/research/cnlab/cost239/"

[46] Sinclair M.C., "Personal Communication"

APPENDIX A

LINEAR PROGRAMMING / SIMPLEX METHOD

A.1. Basic Concepts of Linear Programming

The subject of *linear programming*, sometimes called *linear optimisation*, concerns itself with the following problem : For N independent variables $x_1, \dots, x_N,$ *maximise* the function :

$$z = a_{01}x_1 + a_{02}x_2 + \dots + a_{0N}x_N \qquad \text{(A.1-1)}$$

subject to the primary constraints :

$$x_1 \geq 0, x_2 \geq 0, \dots, x_N \geq 0 \qquad \text{(A.1-2)}$$

and simultaneously subject to $M = m_1 + m_2 + m_3$ additional constraints, m_1 of them of the form :

$$a_{i1}x_1 + a_{i2}x_2 + \dots + a_{iN}x_N \leq b_i \ \ (b_i \geq 0), \quad i = 1, \dots, m_1 \qquad \text{(A.1-3)}$$

m_2 of the form :

$$a_{j1}x_1 + a_{j2}x_2 + \dots + a_{jN}x_N \geq b_j \geq 0, \quad j = m_1 + 1, \dots, m_1 + m_2 \qquad \text{(A.1-4)}$$

and m_3 of the form :

$$a_{k1}x_1 + a_{k2}x_2 + \dots + a_{kN}x_N = b_k \geq 0,$$
$$k = m_1 + m2 + 1, \dots, m_1 + m_2 + m_3 \qquad \text{(A.1-5)}$$

The various a_{ij}'s can have either sign, or be zero. The fact that the b's must all be nonnegative (as indicated by the final inequality in the above three equations) is a matter of convention only, since you can multiply any contrary inequality by -1. There is no particular significance in the number of constraints M being less than, equal to, or greater than the number of unknowns N.

A set of values $x_1, \dots, x_N$ that satisfies the constraints (A.1-2) - (A.1-5) is called a *feasible vector*. That function that is tried to maximised is the *objective function*. The feasible vector that maximises the objective function is the *optimal feasible vector*. An optimal feasible vector can fail to exist for two distinct reasons : (i) there are *no* feasible vectors, i.e., the given constraints are incompatible, or (ii) there is no maximum, i.e., there is a direction in N space where one or more of the variables can be taken to infinity while still satisfying the constraints, giving an unbounded value for the objective function.

The great importance of linear programming can be supported by the following : (i) “nonnegativity” is the usual constraint on any variable x_i that represents the tangible amount of some physical commodity, like guns, butter, dollars, units of vitamin E, food calories, kilowatt hours, mass, etc. Hence, equation (A.1-2). (ii) one is often interested in additive (linear) limitations or bounds posed by man or nature : minimum nutritional requirement, maximum affordable cost, maximum or available labor, or capital, minimum tolerable level of voter approval, etc. Hence, equations (A.1-3) - (A.1-5). (iii) the function that wants one to optimise may be linear, or else may at least be approximated by a linear function - since that is the problem that linear programming can solve. Hence, equation (A.1-1).

The basic concepts of linear optimisation are depicted in fig. (A.1-1), which presents the case of only two independent variables, x_1 and x_2. The linear function z, to be maximised, is represented by its contour lines. Primary constraint require x_1 and x_2 to be positive. Additional constraint may restrict the solution to regions (inequality constraints) or to surfaces (equality constraints). Feasible vectors satisfy all constraints. Feasible basic vectors also lie on the boundary of the allowed region.

Here is a specific example of a problem in linear programming, which has $N = 4$, $m_1 = 2$, $m_2 = m_3 = 1$, hence $M = 4$:

$$\textit{Maximise} : z = x_1 + x_2 + 3x_3 - \frac{1}{2}x_4 \qquad \text{(A.1-6)}$$

with the x’s nonnegative and also with :

$$\begin{aligned} &x_1 + 2x_3 \leq 740 \\ &2x_2 - 7x_4 \leq 0 \\ &x_2 - x_3 + 2x_4 \geq \frac{1}{2} \\ &x_1 + x_2 + x_3 + x_4 = 9 \end{aligned} \qquad \text{(A.1-7)}$$

The answer turns out to be (to 2 decimals) $x_1 = 0$, $x_2 = 3.33$, $x_3 = 4.73$, $x_4 = 0.95$ [36, 37, 40].

Fig. A.1.1 : Basic concepts of linear programming

A.2. Fundamental Theorem of Linear Optimisation

Assuming that we start with a full N-dimensional space of candidate vectors, then we carve away the regions that are eliminated in turn by each posed constraint. Since the constraints are linear, every boundary introduced by this process is a plane, or rather hyperplane. Equality constraints of the form (A.1-5) force the feasible region into hyperplanes of smaller dimension, while inequalities simply divide the then-feasible region into allowed and nonallowed pieces.

When all the constraints are posed, either we are left with some feasible region or else there are no feasible vectors. Since the feasible region is bounded by hyperplanes, it is geometrically a kind of convex polyhedron or simplex. The question arising is whether, if there is a feasible region, the optimal feasible vector can be somewhere in its interior, away from the boundaries. The answer is negative, as the objective function is linear. This means that it always has a nonzero vector gradient, which, in turn, means that we could always increase the objective function by running up the gradient until we hit a boundary wall.

The boundary of any geometrical region has one less dimension than its interior. Therefore, we can now run up the gradient projected into the boundary wall until we reach an edge of that wall. We can then run up that edge, and so on, down through whatever number of dimensions, until we finally arrive at a point, a *vertex* of the original simplex. Since this point has all N of its co-ordinated defined, it must be the solution of N simultaneous equalities drawn from the original set of equalities and inequalities (A.1-2) - (A.1-5).

Points that are feasible vectors and that satisfy N of the original constraints as equalities, are termed *feasible basic vectors*. If N > M, then a feasible basic vector has at least N - M of its components equal to zero, since at least that many of the constraints (A.1-2) will be needed to make up the total of N. Put the other way, at most M components of the feasible basic vector are nonzero. In the example (A.1-6) - (A.1-7), it can be checked that the solution as given satisfies as equalities the last three constraints of (A.1-7) and the constraint $x_1 \geq 0$, for the required total of 4.

If the two preceding paragraphs are put together we obtain the *Fundamental Theorem of Linear Optimisation* : <u>If an optimal vector exists, then there is a feasible basic vector that is optimal</u>.

The importance of the fundamental theorem is that it reduces the optimisation problem to a “combinatorial” problem, that of determining which N constraints (out of the M + N constraints in A.1-2 - A.1-5) should be satisfied by the optimal feasible vector. We have only to keep trying different combinations, and computing the objective function for each trial, until the best is found.

Doing this blindly would take halfway to forever. The *simplex method*, first published by Dantzig in 1948, is a way of organising the procedure so that (i) a series of combinations is tried for which the objective function increases at each step, and (ii) the optimal feasible vector is reached after a number of iterations that is almost always no larger than of order M or N, whichever is larger. An interesting mathematical sidelight is that this second property, although known empirically ever since the simplex method was devised, was not proved to be true until the 1982 work of Stephen Smale [36, 42].

A.3. Simplex Method

A.3.1. Simplex Method for a Restricted Normal Form

A linear programming problem is said to be in *normal form* if it has no constraints in the form (A.1-3) or (A.1-4), but rather only equality constraints of the form (A.1-5) and nonnegativity constraints of the form (A.1-2).

For our purposes, it will be useful to consider an even more restricted set of cases, with this additional property : Each equality constraint of the form (A.1-5) must have at least one variable that has a positive coefficient and *that appears uniquely in that one constraint only*. We can then choose one such variable in each constraint equation, and solve that constraint equation for it. The variables thus chosen are called *left-hand variables*, or *basic variables*, and there are exactly M (= m_3) of them. The remaining N - M variables are called *right-hand variables* or *nonbasic variables*. Obviously, the *restricted normal form* can be achieved only in the case $M \leq N$, so that is the case we will consider.

It may be thought that the restricted normal form is so specialised that it is unlikely to include the linear programming that one wishes to solve. However, this is not true. It will be shown that *any* linear programming problem can be transformed into restricted normal form.

Here is an example of a problem in restricted normal form :

$$\text{Maximise : } z = 2x_2 - 4x_3 \qquad \text{(A.3-1)}$$

with x_1, x_2, x_3 and x_4 all nonnegative and also with :

$$\begin{aligned} x_1 &= 2 - 6x_2 + x_3 \\ x_4 &= 8 + 3x_2 - 4x_3 \end{aligned} \qquad \text{(A.3-2)}$$

This example has N = 4, M = 2; the left-hand variables are x_1 and x_4; the right-hand variables are x_2 and x_3. The objective function (A.1-1) is written so as to depend only on right-hand variables; it should be noted, however, that this is not an actual restriction on objective functions in restricted normal form, since any left-hand variables appearing in the objective function could be eliminated algebraically by use of (A.3-2) or its analogs.

For any problem in restricted normal form, we can instantly read off a feasible basic vector (although not necessarily the *optimal* feasible basic vector). Simply set all right-hand variables equal to zero, and equation (A.3-2) then gives the values of the left-hand variables for which the constraints are satisfied. The idea of the simplex method is to proceed by a series of exchanges. In each exchange, a right-hand variable and a left-hand variable change places. At each stage we maintain a problem in restricted normal form that is equivalent to the original problem.

It is notationally convenient to record the information content of equations (A.3-1) and (A.3-2) in a so-called *tableaux*, as follows :

		x_2	x_3
z	0	2	-4
x_1	2	-6	1
x_4	8	3	-4

(A.3-3)

The *first step* in the simplex method is to examine the top row of the tableaux, which will be called the “z-row”. The columns labelled by right-hand variables will be called “right- columns”. In case that each right-hand variable is increased from its present value of zero, while all the other right-hand variables are left to zero, then the objective function will be increased or decreased, depending on the sign on the entry in the z-row. Since we want to increase the objective function, only right columns, having positive z-row entries, are of interest. In (A.3-3), there is only one such column, whose z-row entry is 2.

The *second step* is to examine the column entries below each z-row entry that was selected by step one. The question is how much can the right-hand variable be increased, before one of the left-hand variables is driven negative, which is not allowed. If the tableaux element in the intersection of the right-hand column and the left-hand variable row is positive, then it posses no restriction : the corresponding left-hand variable will just be driven more and more positive. If all the entries in any right-hand column are positive, then there is no bound on the objective function and we are done with the problem.

If one or more entries below a positive z-row entry are negative, then we have to figure out which such entry first limits the increase of that column right-hand variable. Evidently, the limiting increase is given by dividing the element in the right-hand column (which is called the *pivot element*) into the element in the "constant column" (leftmost column) of the pivot element row. A value that is small in magnitude is not restrictive. The increase in the objective function for this choice of pivot element is then that value multiplied by the z-row entry of that column. This procedure is repeated on all possible right-hand columns to find the pivot element with the largest such increase. That completes our "choice of a pivot element".

In the above example, the only positive z-row entry is 2. There is only one negative entry below it, namely -6, so this is the pivot element. Its constant column entry is 2. This pivot will therefore allow x_2 to be increased by $2 \div |6|$, which results in an increase of the objective function by an amount $(2 \text{ x } 2) \div |6|$.

The *third step* is to do the increase of the selected right-hand variable, thus making it left-hand variable; and simultaneously to modify the left-hand variables, reducing the pivot-row element to zero and thus making it right-hand variable. For our above example, we would begin by solving the pivot row equation for the new left-hand variable x_2 in favour of the old one x_1, namely :

$$x_1 = 2 - 6x_2 + x_3 \quad \rightarrow x_2 = \frac{1}{3} - \frac{1}{6}x_1 + \frac{1}{6}x_3 \qquad \text{(A.3-4)}$$

We then substitute this into the old z-row,

$$z = 2x_2 - 4x_3 = 2\,[\frac{1}{3} - \frac{1}{6}x_1 + \frac{1}{6}x_3] - 4x_3 = \frac{2}{3} - \frac{1}{3}x_1 - \frac{11}{3}x_3 \qquad \text{(A.3-5)}$$

and into all other left-variable rows, in this case only x_4,

$$x_4 = 8 + 3\,[\frac{1}{3} - \frac{1}{6}x_1 + \frac{1}{6}x_3] - 4x_3 = 9 - \frac{1}{2}x_1 - \frac{7}{2}x_3 \qquad \text{(A.3-6)}$$

Equations (A.3-4) - (A.3-6) form the new tableaux :

		x_1	x_3
z	$\frac{2}{3}$	$-\frac{1}{3}$	$-\frac{11}{3}$
x_2	$\frac{1}{3}$	$-\frac{1}{6}$	$\frac{1}{6}$
x_4	9	$-\frac{1}{2}$	$-\frac{7}{2}$

(A.3-7)

The *fourth step* is to go back and repeat the first step, looking for another possible increase of the objective function. We do this as many times as possible, that is, until all the right-hand entries in the z-row are negative, signalling that no further is possible. In the present example, this already occurs in (A.3-7), so the procedure is finished. The answer can now be read from the constant column of the final tableaux. In (A.3-7) it can be seen that the objective function is maximised to a value of $\frac{2}{3}$ for the solution vector $x_2 = \frac{1}{3}$, $x_4 = 9$, $x_1 = x_3 = 0$.

Thus, the procedure that led from (A.3-3) to (A.3-7) could be summarised in the following :

- Locate the pivot element and save it
- Save the whole pivot column
- Replace each row, except the pivot row, by that linear combination of itself and the pivot row which makes its pivot-column entry zero.
- Divide the pivot row by the negative of the pivot.
- Replace the pivot element by the reciprocal of its saved value.
- Replace the rest of the pivot column by its saved values divided by the saved pivot element [36,39].

A.3.2. Writing the General Problem in Restricted Normal Form

Let us consider again the linear programming problem, expressed (A.1-6) - (A.1-7). The first step is to get rid of the inequalities, for example, the first three constraints in (A.1-7). This can be done by adding to the problem so-called *slack* variables which, when their nonnegativity is required, the inequalities are converted to equalities. The slack variables will be denoted as y_i . There will be $m_1 + m_2$ of them. Once they are introduced, they are treated on an equal way with the other variables x_i ; then, at the very end, they are just ignored.

For example, introducing slack variables leaves (A.1-6) unchanged, but turns (A.1.7) into :

$$\begin{aligned} &x_1 + 2x_3 + y_1 = 740 \\ &2x_2 - 7x_4 + y_2 = 0 \\ &x_2 - x_3 + 2x_4 - y_3 = \frac{1}{2} \\ &x_1 + x_2 + x_3 + x_4 = 9 \end{aligned} \qquad \text{(A.3-8)}$$

The sign of the coefficient of the slack variable is determined by which sense of inequality it is replacing.

The second step is to insure that there is a set of M left-hand vectors, so that we can setup a starting tableaux in restricted normal form (in other words, a “feasible basic starting vector” is needed). Thus, some new variables have to be invented. There are M of these, and they are called *artificial variables*; they are denoted by z_i. There is only one artificial variable that will be put into each constraint equation on the following model for the example (A.3-8) :

$$\begin{aligned} &z_1 = 740 - x1 - 2x_3 - y_1 \\ &z_2 = -2x_2 + 7x_4 - y_2 \\ &z_3 = \frac{1}{2} - x_2 + x_3 - 2x_4 + y_3 \\ &z_4 = 9 - x_1 - x_2 - x_3 - x_4 \end{aligned} \qquad \text{(A.3-9)}$$

Now, the example is in restricted normal form.

However, this is not the same problem as (A.3-8) or (A.1-7), unless all the z_i’s are zero. Thus, the problem must be solved in two phases. First phase : the objective function is replaced by a so-called auxiliary objective function :

$$z' \equiv -z_1 - z_2 - z_3 - z_4 = -(749\frac{1}{2} - 2x_1 - 4x_2 - 2x_3 + 4x_4 - y_1 - y_2 + y_3) \qquad \text{(A.4-10)}$$

(where the last equality follows from using (A.3-9). We now perform the simplex method on the auxiliary objective function (A.4-10) with the constraints (A.3-9).

Obviously, the auxiliary objective function will be maximised for nonnegativity z_i's if all the z_i's are zero. We therefore expect the simplex method in this first phase to produce a set of left-hand variables drawn from the x_i's and y_i's only, with all the z_i's being right-hand variables. Then, we cross out the z_i's, leaving a problem involving only x_i's and y_i's in restricted normal form. In other words, the first phase produces an initial feasible basic vector. Second phase : the problem produced by the first phase is solved, using the original objective function, not the auxiliary.

In case that the first phase doesn't produce zero values for all the z_i's, then there is no initial feasible basic vector, i.e., the constraints given are inconsistent among themselves.

Here is how to translate into tableaux format the information needed for both the first and second phases of the overall method. As before, the underlying problem to be solved is as posed in equations (A.1-6) and (A.1-7).

		x_1	x_2	x_3	x_4	y_1	y_2	y_3
z	0	1	1	3	$-\frac{1}{2}$	0	0	0
z_1	740	-1	0	-2	0	-1	0	0
z_2	0	0	-2	0	7	0	-1	0
z_3	$\frac{1}{2}$	0	-1	1	-2	0	0	1
z_4	9	-1	-1	-1	-1	0	0	0
z'	$-749\frac{1}{2}$	2	4	2	-4	1	1	-1

(A.3-11)

This is as daunting as it may, at first sight, appear. The table entries inside the box of doubled lines are no more than the coefficients of the original problem (A.1-6) - (A.1-7) organised into a tabular form. In fact, these entries, along with the values of N, M, m_1, m_2 and m_3, are the only input that is needed by the simplex method routine. The columns under the slack variables y_i simply record whether each of the M constraints is of the form $\leq, \geq$, or =; this is redundant information with the values m_1, m_2, m_3, as long as we are sure to enter the rows of the tableaux in the correct respective order.

The coefficients of the auxiliary objective function (bottom row) are just the negatives of the columns sums of the rows above, so these are easily calculated automatically.

The output from a simplex routine will be : (i) a flag telling whether a finite solution, no solution, or an unbounded solution was found, and (ii) an updated tableaux. The output tableaux that derives from (A.3-11) is :

		x_1	y_2	y_3	.....
z	17.03	-0.95	-0.5	-1.05	.....
x_2	3.33	-0.35	-0.15	0.35	.....
x_3	4.73	-0.55	0.05	-0.45	.....
x_4	0.95	-0.1	0.1	0.1	.....
y_1	730.55	0.1	-0.1	0.9	.....

(A.3-12)

By counting the x_i's and y_i's it will be seen that there are M + 1 rows (including the z-row) in both the input and the output tableaux, but only N + 1 - m_3 columns of the output tableau (including the constant column) contain any useful information, the other columns belonging to now-discarded artificial variables. In the output, the first numerical column contains the solution vector, along with the maximum value of the objective function. Where a slack variable (y_i) appears on the left, the corresponding value is the amount by which its inequality is safely satisfied. Variables that are not left-hand variables, in the output tableau have zero values. Slack variables with zero values represent constraints that are satisfied as equalities" [36].

APPENDIX B

k-HOP ROUTING - EVALUATION OF THE NUMBER OF PATHS

B.1. Paths

Paths are an essential element of a network. By the term “path” is meant a series of links that connect a pair of nodes. A path may be consisted of more than one links. In case that only one link is used (the pair of nodes is directly connected and the traffic between them is routed via this link) we have a “*direct path*” or a “*one-hop path*”. Similarly, we may have a *two* or *three-hop path* in case that the traffic between this pair of nodes is routed through one or two, respectively, intermediates nodes.

It is clear that if only direct paths are used, there is a unique route between the source and the destination node (this is the link that connects them). On the other hand, in case that a routing algorithm of two, three or more hops is used, there are more than one ways between them. Then, it is not necessary that the traffic between the nodes should follow only one specific route (the traffic may be split between two or more routes that may be disjoint or may be not).

As in the case of links, there is a need to introduce in the LP problem a new group of variables that will represent those paths : y_i, i = 0,1,2, ... Number of Paths. The development of a methodology of evaluating the number of paths in the network, depending on the number of hops used, can be done by a careful examination of the fig. 3.1 five-nodes network.

B.2. Direct Paths

It is obvious that if only direct paths are used, the total number of paths in the network will be equal to the number of links, that is :

$$N_{tot} = N_{Dir} = \frac{N(N-1)}{2}, \qquad \text{(B.2-1)}$$

where N is the number of nodes.

It should be stressed that all the calculations are valid *only* for complete mesh networks (full interconnection between nodes).

B.3. 2-Hop Paths

In the 2-hop routing case, for every pair of nodes, apart from the direct link, N - 2 paths can be used, as N - 2 nodes (all the network nodes, apart from the source and the destination) can be used as intermediates. Therefore, the number of 2-hop paths will be :

$$N_{2\text{-Hop}} = N_{Dir} * (N - 2) = \frac{N(N-1)}{2} (N - 2) \qquad \text{(B.3-1)}$$

and the total number of available paths in the network :

$$N_{tot} = N_{Dir} + N_{2\text{-Hop}} \qquad \text{(B.3-2)}$$

Thus, in the example network of fig. 3.1, the 2-hop paths between Nodes 0 and 1 will be :

- Node 0 - Node 2 - Node 1
- Node 0 - Node 3 - Node 1
- Node 0 - Node 4 - Node 1

B.4. 3-Hop Paths

Likewise, in the 3-hop routing case, for every 2-hop path between a pair of nodes, N - 3 paths can be used, as N - 3 nodes (all the network nodes, apart from the source, the destination and the first intermediate node) can be used as second intermediates. Therefore, the number of 3-hop paths will be :

$$N_{3\text{-Hop}} = N_{2\text{-Hop}} * (N - 3) = \frac{N(N-1)}{2} (N - 2)(N - 3) \qquad \text{(B.4-1)}$$

and the total number of available paths in the network :

$$N_{tot} = N_{Dir} + N_{2\text{-Hop}} + N_{3\text{-Hop}} \qquad \text{(B.4-2)}$$

Thus, in the example network of fig. 3.1, the 3-hop paths between Nodes 0 and 1 will be :

- Node 0 - Node 2 - Node 3 - Node 1
- Node 0 - Node 2 - Node 4 - Node 1
- Node 0 - Node 3 - Node 2 - Node 1
- Node 0 - Node 3 - Node 4 - Node 1
- Node 0 - Node 4 - Node 2 - Node 1
- Node 0 - Node 4 - Node 3 - Node 1

Now, it is clear, that in the general case of k-hop routing, the number of k-hop paths will be :

$$N_{\text{k-Hop}} = N\,(N - 1)\,(N - 2)\,...\,(N - k)\,/\,2 \qquad \text{(B.4-3)}$$

whereas the total available number of paths in the network :

$$N_{\text{tot}} = \sum_{i=1}^{k} N_{i-Hop} \qquad \text{(B.4-4)}$$

APPENDIX C

VIRTUAL TOPOLOGY DESIGN : RESULTS

C.1. The Objective Function and the set of Constraints for the 5-nodes Network of fig.3.1

The list following includes, apart from the objective function, *all* the (link & path) constraints set for the formulation of an LP problem corresponding to the 5-nodes network of fig. 3.1 (section 3.2.1) :

```
min: 916.4 x0 + 873.241 x1 + 1143.96 x2 + 522.726 x3 + 784.307 x4 + 318.057 x5 +
805.355 x6 + 1096.4 x7 + 368.399 x8 + 1112.68 x9;

x0 + x1 + x2 + x3 >= 22;
x0 + x4 + x5 + x6 >= 19.2;
x1 + x4 + x7 + x8 >= 23.7;
x2 + x5 + x7 + x9 >= 23.5;
x3 + x6 + x8 + x9 >= 16;

x1 + x2 + x3 + x4 + x5 + x6 >= 31.2;
x0 + x2 + x3 + x4 + x7 + x8 >= 33.7;
x0 + x1 + x3 + x5 + x7 + x9 >= 30.5;
x0 + x1 + x2 + x6 + x8 + x9 >= 31;
x0 + x1 + x5 + x6 + x7 + x8 >= 28.5;
x0 + x2 + x4 + x6 + x7 + x9 >= 38.7;
x0 + x3 + x4 + x5 + x8 + x9 >= 25.2;
x1 + x2 + x4 + x5 + x8 + x9 >= 30.2;
x1 + x3 + x4 + x6 + x7 + x9 >= 35.7;
x2 + x3 + x5 + x6 + x7 + x8 >= 28.5;

x10 + x11 + x12 + x13 = 5;
x14 + x15 + x16 + x17 = 6;
x18 + x19 + x20 + x21 = 7.5;
x22 + x23 + x24 + x25 = 3.5;
x26 + x27 + x28 + x29 = 7.2;
```

x30 + x31 + x32 + x33 = 2;
x34 + x35 + x36 + x37 = 5;
x38 + x39 + x40 + x41 = 8.5;
x42 + x43 + x44 + x45 = 2;
x46 + x47 + x48 + x49 = 5.5;

x10 + x15 + x19 + x23 + x27 + x31 + x35 = x0;
x11 + x14 + x20 + x24 + x27 + x39 + x43 = x1;
x12 + x16 + x18 + x25 + x31 + x39 + x47 = x2;
x13 + x17 + x21 + x22 + x35 + x43 + x47 = x3;
x11 + x15 + x26 + x32 + x36 + x40 + x44 = x4;
x12 + x19 + x28 + x30 + x37 + x40 + x48 = x5;
x13 + x23 + x29 + x33 + x34 + x44 + x48 = x6;
x16 + x20 + x28 + x32 + x38 + x45 + x49 = x7;
x17 + x24 + x29 + x36 + x41 + x42 + x49 = x8;
x21 + x25 + x33 + x37 + x41 + x45 + x46 = x9;

C.2. Traffic Matrix and the Final Optimum Capacity found for EON

Node	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
0	0	0.59	0.25	1.53	23.3	0.34	0.08	1.57	0.08	1.44	0.07	0.08	0.51	0.59	4.32	1.78	1.1	1.61	0.34	1.57	Vienna
1	0.51	0	0.59	17.2	9.32	0.68	0.34	1.82	2.54	14.3	0.34	0.68	1.95	0.93	1.86	6.44	0.29	0.17	0.17	1.82	Brussels
2	0.34	0.68	0	1.93	5.93	0.17	0.17	0.43	0.17	1.27	3.56	0.17	0.68	5.17	0.76	3.73	0.07	0.07	0.07	0.43	Copenhagen
3	1.36	17.2	1.61	0	23.3	1.27	0.93	8.6	1.96	6.52	0.68	2.04	12.3	1.78	16.7	22.5	0.74	0.42	1.37	8.6	Paris
4	26.1	9.32	5.59	26.3	0	7.12	1.1	11.6	2.29	19.7	1.69	2.54	9.79	14.2	24.9	25.4	2.66	4.74	5.63	11.6	Berlin
5	0.42	0.59	0.17	1.27	4.66	0	0.08	0.93	0.08	0.59	0.08	0	0.17	0.34	0.68	2.8	0.32	0.08	0	0.93	Athens
6	0	0.25	0.17	0.76	0	0	0	0.13	0	0.42	0	0	0.93	0.08	0.17	0.76	0.02	0	0	0.13	Dublin
7	1.53	1.7	0.34	8.01	8.81	0.81	0.17	0	0.21	1.23	0.17	0.3	2.04	0.47	5.76	3.82	0.71	0.38	0.26	0	Rome
8	0.17	3.22	0.25	2.71	2.8	0.08	0.08	0.34	0	0.51	0.08	0.85	0.25	0.17	0.42	0.85	0.06	0	0	0.34	Luxembourg
9	1.27	14.6	1.27	6.52	20.1	0.59	0.59	1.4	0.51	0	0.85	0.59	2.2	1.27	2.46	11.5	0.34	0.34	0.25	1.4	Amsterdam
10	0.17	0.34	3.81	0.85	1.95	0.08	0.08	0.21	0.08	0.08	0	0.08	0.51	7.63	0.42	3.39	0.08	0.08	0.08	0.21	Oslo
11	0.08	0.51	0.17	3.13	1.44	0	0.08	0.26	0.17	0.51	0.08	0	1.95	0.25	0.93	1.86	0.08	0	0	0.26	Lisbon
12	0.51	2.12	0.68	9.6	9.79	0.25	0.59	2.16	0.41	2.37	0.42	2.2	0	1.27	2.54	4.54	0.35	0.08	0.65	2.16	Madrid
13	0.68	0.93	6.69	3.06	14.2	0.42	0.08	0.6	0.23	1.02	8.05	0.33	1.27	0	1.27	2.31	0.26	0.34	1.2	0.6	Stockholm
14	5.08	2.2	0.76	20.4	28.8	0.76	0.25	9.15	0.51	2.97	0.42	4.66	4.32	1.27	0	6.69	0.93	0.85	0.34	9.15	Zurich
15	1.19	5.85	3.39	19.1	19.1	2.29	0.76	4.58	1.05	9.74	2.8	1.11	4.54	2.31	6.27	0	0.44	0.51	0.93	4.58	London
16	0.93	0.08	0	0.25	3.22	0	0	0.47	0	0.17	0	0	0	0.17	0.51	0.25	0	0.17	0.08	0.47	Zagreb
17	1.19	0.17	0.08	0.34	2.63	0.08	0	0.26	0	0.25	0	0	0.08	0.17	0.42	0.34	0.07	0	0.42	0.26	Prague
18	0.68	0.93	6.69	1.37	5.63	0.42	0.08	0.6	0.1	1.02	8.05	0.17	0.65	1.2	1.27	0.93	0.17	0.34	0	0.6	Moscow
19	1.53	1.7	0.34	8.01	8.81	0.81	0.17	0	0.21	1.23	0.17	0.3	2.04	0.47	5.76	3.82	0.71	0.38	0.26	0	Milan

Table C.1 : Traffic Matrix of European Optical Network

Node	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	-	1.1	0.59	2.89	49.4	0.76	0.08	3.1	0.25	2.71	0.34	0.16	1.02	1.27	9.4	2.97	2.03	2.8	1.02	3.1
1	-	-	1.27	34.4	18.64	1.27	0.59	3.52	5.76	28.89	0.68	1.19	4.07	1.86	4.06	12.29	0.37	0.34	1.1	3.52
2	-	-	-	3.14	11.52	0.34	0.34	0.77	0.42	2.54	7.37	0.34	1.36	11.86	1.52	7.12	0.08	0.16	6.77	0.77
3	-	-	-	-	49.57	2.54	1.69	16.61	4.67	13.04	1.53	5.17	21.89	4.84	37.11	41.51	0.99	0.76	2.74	16.61
4	-	-	-	-	-	11.78	1.1	20.42	5.09	39.91	3.64	3.98	18.58	28.44	53.63	44.48	5.88	7.37	11.26	20.42
5	-	-	-	-	-	-	0.08	1.74	0.16	1.18	0.16	0	0.42	0.76	1.44	5.09	0.32	0.16	0.42	1.74
6	-	-	-	-	-	-	-	0.3	0.08	1.01	0.08	0.08	1.52	0.16	0.42	1.52	0.02	0	0.08	0.3
7	-	-	-	-	-	-	-	-	0.55	2.63	0.38	0.56	4.2	1.07	14.91	8.4	1.18	0.64	0.86	0
8	-	-	-	-	-	-	-	-	-	1.02	0.16	1.02	0.66	0.4	0.93	1.9	0.06	0	0.1	0.55
9	-	-	-	-	-	-	-	-	-	-	0.93	1.1	4.57	2.29	5.43	21.26	0.51	0.59	1.27	2.63
10	-	-	-	-	-	-	-	-	-	-	-	0.16	0.93	15.68	0.84	6.19	0.08	0.08	8.13	0.38
11	-	-	-	-	-	-	-	-	-	-	-	-	4.15	0.58	5.59	2.97	0.08	0	0.17	0.56
12	-	-	-	-	-	-	-	-	-	-	-	-	-	2.54	6.86	9.08	0.35	0.16	1.3	4.2
13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2.54	4.62	0.43	0.51	2.4	1.07
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	12.96	1.44	1.27	1.61	14.91
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.69	0.85	1.86	8.4
16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.24	0.25	1.18
17	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.76	0.64
18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.86
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table C.2 : "Optimal" capacities of links for the EON network

C.3. List of the 2-Hop Paths Arising When Limitations on the Traffic and the Length of Links are set

The following list includes, apart from the 2-hop paths, the index of the capacity variable and the traffic caried, corresponding to each path :

x299 : Vienna-London-Dublin - Traffic : 0.08
x398 : Vienna-Milan-Lisbon - Traffic : 0.16
x412 : Vienna-Zurich-Madrid - Traffic : 1.02
x621 : Brussels-Zurich-Athens - Traffic : 1.27
x725 : Brussels-Paris-Lisbon - Traffic : 1.19
x859 : Brussels-Berlin-Moscow - Traffic : 1.1
x946 : Copenhagen-Zagreb-Athens - Traffic : 0.34
x1048 : Copenhagen-Paris-Lisbon - Traffic : 0.34
x1066 : Copenhagen-Brussels-Madrid - Traffic : 1.36
x1253 : Paris-Milan-Athens - Traffic : 2.54
x1390 : Paris-Copenhagen-Stockholm - Traffic : 4.84
x1486 : Paris-Berlin-Moscow - Traffic : 2.74
x1535 : Berlin-Zagreb-Athens - Traffic : 11.78
x1642 : Berlin-Luxembourg-Lisbon - Traffic : 3.98
x1661 : Berlin-Luxembourg-Madrid - Traffic : 19.58
x1818 : Athens-Zurich-Dublin - Traffic : 0.08

```
x1856 : Athens-Zurich-Luxembourg - Traffic : 0.16
x1877 : Athens-Zagreb-Amsterdam - Traffic : 1.18
x1896 : Athens-Zagreb-Oslo - Traffic : 0.16
x1990 : Athens-Zurich-London - Traffic : 5.09
x2030 : Athens-Zagreb-Prague - Traffic : 0.16
x2257 : Dublin-London-Zagreb - Traffic : 0.02
x2283 : Dublin-Copenhagen-Moscow - Traffic : 0.08
x2524 : Rome-Zagreb-Moscow - Traffic : 0.86
x2722 : Luxembourg-Berlin-Moscow - Traffic : 0.1
x2778 : Amsterdam-Paris-Lisbon - Traffic : 1.1
x2912 : Amsterdam-Berlin-Moscow - Traffic : 1.27
x2959 : Oslo-London-Lisbon - Traffic : 0.16
x2968 : Oslo-Paris-Madrid - Traffic : 0.93
x3149 : Lisbon-London-Stockholm - Traffic : 0.58
x3166 : Lisbon-Madrid-Zurich - Traffic : 5.59
x3210 : Lisbon-Milan-Zagreb - Traffic : 0.08
x3236 : Lisbon-Athens-Moscow - Traffic : 0.17
x3270 : Madrid-Brussels-Stockholm - Traffic : 2.54
x3358 : Madrid-Zurich-Prague - Traffic : 0.16
x3379 : Madrid-Zagreb-Moscow - Traffic : 1.3
x3513 : Stockholm-Prague-Milan - Traffic : 1.07
x3589 : Zurich-Prague-Moscow - Traffic : 1.61
x3651 : London-Copenhagen-Moscow - Traffic : 1.86
x3782 : Moscow-Vienna-Milan - Traffic : 0.86
--------------------------------------------------
Number of links used in the network : 146
Number of two-hop paths used : 40

Links' Cost : 876678
Nodes cost : 2151.04
Network cost (Value of objective function) : 878829
```

C.4. The List of the 2-Hop Paths Selected for the Backup Connections of EON

```
x3807 : Vienna-Luxembourg-Brussels : 1.1
x3822 : Vienna-Berlin-Copenhagen : 0.59
x3845 : Vienna-Luxembourg-Paris : 2.89
x3873 : Vienna-Prague-Berlin : 49.4
x3891 : Vienna-Zagreb-Athens : 0.76
x3909 : Vienna-London-Dublin : 0.08
x3932 : Vienna-Milan-Rome : 3.1
x3949 : Vienna-Prague-Luxembourg : 0.25
```

x3968 : Vienna-Prague-Amsterdam : 2.71
x3975 : Vienna-Berlin-Oslo : 0.34
x4003 : Vienna-Zurich-Lisbon : 0.16
x4022 : Vienna-Zurich-Madrid : 1.02
x4032 : Vienna-Berlin-Stockholm : 1.27
x4063 : Vienna-Prague-Zurich : 9.4
x4082 : Vienna-Prague-London : 2.97
x4101 : Vienna-Prague-Zagreb : 2.03
x4108 : Vienna-Berlin-Prague : 2.8
x4127 : Vienna-Berlin-Moscow : 1.02
x4156 : Vienna-Zurich-Milan : 3.1
x4169 : Brussels-Amsterdam-Copenhagen : 1.27
x4187 : Brussels-Luxembourg-Paris : 34.4
x4206 : Brussels-Luxembourg-Berlin : 18.64
x4225 : Brussels-Luxembourg-Athens : 1.27
x4251 : Brussels-London-Dublin : 0.59
x4274 : Brussels-Milan-Rome : 3.52
x4283 : Brussels-Amsterdam-Luxembourg : 5.76
x4302 : Brussels-Luxembourg-Amsterdam : 28.89
x4322 : Brussels-Amsterdam-Oslo : 0.68
x4335 : Brussels-Paris-Lisbon : 1.19
x4354 : Brussels-Paris-Madrid : 4.07
x4372 : Brussels-Copenhagen-Stockholm : 1.86
x4397 : Brussels-Luxembourg-Zurich : 4.06
x4417 : Brussels-Amsterdam-London : 12.29
x4435 : Brussels-Luxembourg-Zagreb : 0.37
x4454 : Brussels-Luxembourg-Prague : 0.34
x4469 : Brussels-Berlin-Moscow : 1.1
x4492 : Brussels-Luxembourg-Milan : 3.52
x4505 : Copenhagen-Brussels-Paris : 3.14
x4538 : Copenhagen-Prague-Berlin : 11.52
x4556 : Copenhagen-Zagreb-Athens : 0.34
x4574 : Copenhagen-London-Dublin : 0.34
x4595 : Copenhagen-Prague-Rome : 0.77
x4606 : Copenhagen-Amsterdam-Luxembourg : 0.42
x4619 : Copenhagen-Brussels-Amsterdam : 2.54
x4648 : Copenhagen-Stockholm-Oslo : 7.37
x4658 : Copenhagen-Paris-Lisbon : 0.34
x4676 : Copenhagen-Brussels-Madrid : 1.36
x4703 : Copenhagen-Oslo-Stockholm : 11.86
x4728 : Copenhagen-Prague-Zurich : 1.52
x4740 : Copenhagen-Amsterdam-London : 7.12
x4754 : Copenhagen-Berlin-Zagreb : 0.08

```
x4773 : Copenhagen-Berlin-Prague : 0.16
x4792 : Copenhagen-Berlin-Moscow : 6.77
x4821 : Copenhagen-Zurich-Milan : 0.77
x4833 : Paris-Luxembourg-Berlin : 49.57
x4863 : Paris-Milan-Athens : 2.54
x4878 : Paris-London-Dublin : 1.69
x4901 : Paris-Milan-Rome : 16.61
x4904 : Paris-Brussels-Luxembourg : 4.67
x4923 : Paris-Brussels-Amsterdam : 13.04
x4949 : Paris-Amsterdam-Oslo : 1.53
x4970 : Paris-Madrid-Lisbon : 5.17
x4980 : Paris-Brussels-Madrid : 21.89
x5000 : Paris-Copenhagen-Stockholm : 4.84
x5024 : Paris-Luxembourg-Zurich : 37.11
x5037 : Paris-Brussels-London : 41.51
x5068 : Paris-Zurich-Zagreb : 0.99
x5081 : Paris-Luxembourg-Prague : 0.76
x5096 : Paris-Berlin-Moscow : 2.74
x5125 : Paris-Zurich-Milan : 16.61
x5145 : Berlin-Zagreb-Athens : 11.78
x5157 : Berlin-Amsterdam-Dublin : 1.1
x5184 : Berlin-Prague-Rome : 20.42
x5203 : Berlin-Prague-Luxembourg : 5.09
x5209 : Berlin-Copenhagen-Amsterdam : 39.91
x5228 : Berlin-Copenhagen-Oslo : 3.64
x5252 : Berlin-Luxembourg-Lisbon : 3.98
x5271 : Berlin-Luxembourg-Madrid : 19.58
x5285 : Berlin-Copenhagen-Stockholm : 28.44
x5317 : Berlin-Prague-Zurich : 53.63
x5320 : Berlin-London : 44.48
x5340 : Berlin-Vienna-Zagreb : 5.88
x5359 : Berlin-Vienna-Prague : 7.37
x5380 : Berlin-Copenhagen-Moscow : 11.26
x5413 : Berlin-Prague-Milan : 20.42
x5428 : Athens-Zurich-Dublin : 0.08
x5449 : Athens-Zagreb-Rome : 1.74
x5466 : Athens-Zurich-Luxembourg : 0.16
x5488 : Athens-Prague-Amsterdam : 1.18
x5496 : Athens-Berlin-Oslo : 0.16
x5536 : Athens-Rome-Madrid : 0.42
x5563 : Athens-Zagreb-Stockholm : 0.76
x5585 : Athens-Milan-Zurich : 1.44
x5588 : Athens-Brussels-London : 5.09
```

```
x5606 : Athens-Vienna-Zagreb : 0.32
x5640 : Athens-Zagreb-Prague : 0.16
x5659 : Athens-Zagreb-Moscow : 0.42
x5669 : Athens-Rome-Milan : 1.74
x5685 : Dublin-Paris-Rome : 0.3
x5702 : Dublin-Brussels-Luxembourg : 0.08
x5733 : Dublin-London-Amsterdam : 1.01
x5752 : Dublin-London-Oslo : 0.08
x5768 : Dublin-Madrid-Lisbon : 0.08
x5780 : Dublin-Paris-Madrid : 1.52
x5805 : Dublin-Oslo-Stockholm : 0.16
x5816 : Dublin-Brussels-Zurich : 0.42
x5837 : Dublin-Paris-London : 1.52
x5867 : Dublin-London-Zagreb : 0.02
x5893 : Dublin-Copenhagen-Moscow : 0.08
x5913 : Dublin-Paris-Milan : 0.3
x5946 : Rome-Milan-Luxembourg : 0.55
x5960 : Rome-Zurich-Amsterdam : 2.63
x5969 : Rome-Copenhagen-Oslo : 0.38
x5996 : Rome-Madrid-Lisbon : 0.56
x6022 : Rome-Milan-Madrid : 4.2
x6028 : Rome-Berlin-Stockholm : 1.07
x6060 : Rome-Milan-Zurich : 14.91
x6079 : Rome-Milan-London : 8.4
x6081 : Rome-Vienna-Zagreb : 1.18
x6117 : Rome-Milan-Prague : 0.64
x6134 : Rome-Zagreb-Moscow : 0.86
x6158 : Luxembourg-Brussels-Amsterdam : 1.02
x6184 : Luxembourg-Amsterdam-Oslo : 0.16
x6198 : Luxembourg-Paris-Lisbon : 1.02
x6217 : Luxembourg-Paris-Madrid : 0.66
x6235 : Luxembourg-Copenhagen-Stockholm : 0.4
x6253 : Luxembourg-Brussels-Zurich : 0.93
x6272 : Luxembourg-Brussels-London : 1.9
x6290 : Luxembourg-Vienna-Zagreb : 0.06
x6332 : Luxembourg-Berlin-Moscow : 0.1
x6360 : Luxembourg-Zurich-Milan : 0.55
x6368 : Amsterdam-Copenhagen-Oslo : 0.93
x6388 : Amsterdam-Paris-Lisbon : 1.1
x6407 : Amsterdam-Paris-Madrid : 4.57
x6425 : Amsterdam-Copenhagen-Stockholm : 2.29
x6450 : Amsterdam-Luxembourg-Zurich : 5.43
x6462 : Amsterdam-Brussels-London : 21.26
```

x6495 : Amsterdam-Prague-Zagreb : 0.51
x6507 : Amsterdam-Luxembourg-Prague : 0.59
x6522 : Amsterdam-Berlin-Moscow : 1.27
x6545 : Amsterdam-Luxembourg-Milan : 2.63
x6569 : Oslo-London-Lisbon : 0.16
x6578 : Oslo-Paris-Madrid : 0.93
x6596 : Oslo-Copenhagen-Stockholm : 15.68
x6615 : Oslo-Copenhagen-Zurich : 0.84
x6641 : Oslo-Amsterdam-London : 6.19
x6655 : Oslo-Berlin-Zagreb : 0.08
x6672 : Oslo-Copenhagen-Prague : 0.08
x6701 : Oslo-Stockholm-Moscow : 8.13
x6721 : Oslo-Zurich-Milan : 0.38
x6730 : Lisbon-Paris-Madrid : 4.15
x6755 : Lisbon-Amsterdam-Stockholm : 0.58
x6776 : Lisbon-Madrid-Zurich : 5.59
x6787 : Lisbon-Paris-London : 2.97
x6820 : Lisbon-Milan-Zagreb : 0.08
x6857 : Lisbon-Prague-Moscow : 0.17
x6871 : Lisbon-Madrid-Milan : 0.56
x6881 : Madrid-Copenhagen-Stockholm : 2.54
x6915 : Madrid-Milan-Zurich : 6.86
x6920 : Madrid-Paris-London : 9.08
x6953 : Madrid-Milan-Zagreb : 0.35
x6968 : Madrid-Zurich-Prague : 0.16
x6990 : Madrid-Prague-Moscow : 1.3
x7006 : Madrid-Zurich-Milan : 4.2
x7027 : Stockholm-Prague-Zurich : 2.54
x7033 : Stockholm-Copenhagen-London : 4.62
x7050 : Stockholm-Vienna-Zagreb : 0.43
x7073 : Stockholm-Berlin-Prague : 0.51
x7098 : Stockholm-Oslo-Moscow : 2.4
x7123 : Stockholm-Prague-Milan : 1.07
x7127 : Zurich-Brussels-London : 12.96
x7145 : Zurich-Vienna-Zagreb : 1.44
x7181 : Zurich-Milan-Prague : 1.27
x7199 : Zurich-Prague-Moscow : 1.61
x7210 : Zurich-Luxembourg-Milan : 14.91
x7229 : London-Luxembourg-Zagreb : 0.69
x7249 : London-Amsterdam-Prague : 0.85
x7268 : London-Amsterdam-Moscow : 1.86
x7279 : London-Brussels-Milan : 8.4
x7297 : Zagreb-Vienna-Prague : 0.24

```
x7316 : Zagreb-Vienna-Moscow : 0.25
x7349 : Zagreb-Zurich-Milan : 1.18
x7358 : Prague-Berlin-Moscow : 0.76
x7387 : Prague-Zurich-Milan : 0.64
x7392 : Moscow-Vienna-Milan : 0.86


----------------------------------------------------


Number of links used in the network : 188

Number of primary direct paths used : 184
Number of primary two-hop paths used : 1

Number of backup direct paths used : 1
Number of backup two-hop paths used : 184

Links' Cost : 1.91476e+06
Primary Paths' Cost : 875234
Backup Paths' Cost : 1.03952e+06
Nodes cost : 5976.18
Network cost : 1.92073e+06

Total Links' Capacity : 2988.09
Total Primary Paths' Capacity : 996.03
Total Backup Paths' Capacity : 996.03
```

APPENDIX D

ROUTING & WAVELENGTH ASSIGNMENT : RESULTS

D.1. List of 4-Shortest Paths (and the Corresponding Length) for all Network Node Pairs

This list contains the 4 shortest paths for each one of the 36 existing source-destination pairs the 9-nodes version of EON. The length of the paths and the wavelength variables corresponding to them (in this case there are 4 wavelengths per path) are also included :

Paris - London : 1.94258 (x17-20)

Paris - Brussels - London : 3.31444 (x21-24)

Paris - Brussels - Amsterdam - London : 4.45918 (x25-28)

Paris - Brussels - Luxembourg - Amsterdam - London : 5.60286 (x29-32)

Paris - Brussels - Berlin : 5.59706 (x33-36)

Paris - Brussels - Amsterdam - Berlin : 6.25238 (x37-40)

Paris - Zurich - Berlin : 7.03547 (x41-44)

Paris - Zurich - Prague - Berlin : 7.04859 (x45-48)

Paris - Milan : 3.70034 (x49-52)

Paris - Zurich - Milan : 4.1874 (x53-56)

Paris - Brussels - Luxembourg - Milan : 5.12016 (x57-60)

Paris - Brussels - Amsterdam - Luxembourg - Milan : 7.05414 (x61-64)

Paris - Brussels : 1.58028 (x65-68)

Paris - London - Brussels : 3.67673 (x69-72)

Paris - London - Amsterdam - Brussels : 4.82147 (x73-76)

Paris - London - Amsterdam - Luxembourg - Brussels : 5.96515 (x77-80)

Paris - Brussels - Amsterdam : 2.75774 (x81-84)

Paris - London - Amsterdam : 3.64402 (x85-88)

Paris - Brussels - Luxembourg - Amsterdam : 3.90142 (x89-92)

Paris - London - Brussels - Amsterdam : 4.85419 (x93-96)

Paris - Zurich - Prague : 5.28993 (x97-100)

Paris - Milan - Prague : 6.54445 (x101-104)

Paris - Milan - Zurich - Prague : 6.70381 (x105-108)

Paris - Zurich - Milan - Prague : 7.03152 (x109-112)

Paris - Zurich : 3.23693 (x113-116)

Paris - Milan - Zurich : 4.65081 (x117-120)

Paris - Brussels - Luxembourg - Milan - Zurich : 6.07063 (x121-124)

Paris - Milan - Prague - Zurich : 8.59745 (x125-128)

Paris - Brussels - Luxembourg : 2.36259 (x129-132)

Paris - Brussels - Amsterdam - Luxembourg : 4.29657 (x133-136)

Paris - London - Brussels - Luxembourg : 4.45904 (x137-140)

Paris - London - Amsterdam - Luxembourg : 5.18285 (x141-144)

London - Amsterdam - Berlin : 5.19608 (x145-148)

London - Brussels - Berlin : 5.75093 (x149-152)

London - Brussels - Amsterdam - Berlin : 6.40625 (x153-156)

London - Amsterdam - Brussels - Berlin : 6.89567 (x157-160)

London - Brussels - Luxembourg - Milan : 5.27403 (x161-164)

London - Paris - Milan : 5.64291 (x165-168)

London - Amsterdam - Luxembourg - Milan : 5.99784 (x169-172)

London - Paris - Zurich - Milan : 6.12998 (x173-176)

London - Brussels : 1.73416 (x177-180)

London - Amsterdam - Brussels : 2.8789 (x181-184)

London - Paris - Brussels : 3.52286 (x185-188)

London - Amsterdam - Luxembourg - Brussels : 4.02258 (x189-192)

London - Amsterdam : 1.70144 (x193-196)

London - Brussels - Amsterdam : 2.91161 (x197-200)

London - Brussels - Luxembourg - Amsterdam : 4.05529 (x201-204)

London - Paris - Brussels - Amsterdam : 4.70031 (x205-208)

London - Amsterdam - Berlin - Prague : 6.95474 (x209-212)

London - Paris - Zurich - Prague : 7.2325 (x213-216)

London - Brussels - Berlin - Prague : 7.5096 (x217-220)

London - Brussels - Luxembourg - Milan - Prague : 8.11815 (x221-224)

London - Paris - Zurich : 5.1795 (x225-228)

London - Brussels - Luxembourg - Milan - Zurich : 6.22451 (x229-232)

London - Brussels - Paris - Zurich : 6.55137 (x233-236)

London - Paris - Milan - Zurich : 6.59339 (x237-240)

London - Brussels - Luxembourg : 2.51646 (x241-244)

London - Amsterdam - Luxembourg : 3.24027 (x245-248)

London - Amsterdam - Brussels - Luxembourg : 3.6612 (x249-252)

London - Paris - Brussels - Luxembourg : 4.30516 (x253-256)

Berlin - Prague - Milan : 4.60278 (x257-260)

Berlin - Zurich - Milan : 4.74901 (x261-264)

Berlin - Prague - Zurich - Milan : 4.76214 (x265-268)

Berlin - Brussels - Luxembourg - Milan : 7.55665 (x269-272)

Berlin - Brussels : 4.01678 (x273-276)

Berlin - Amsterdam - Brussels : 4.67209 (x277-280)

Berlin - Amsterdam - Luxembourg - Brussels : 5.81577 (x281-284)

Berlin - Amsterdam - London - Brussels : 6.93024 (x285-288)

Berlin - Amsterdam : 3.49464 (x289-292)

Berlin - Brussels - Amsterdam : 5.19423 (x293-296)

Berlin - Brussels - Luxembourg - Amsterdam : 6.33791 (x297-300)

Berlin - Brussels - London - Amsterdam : 7.45237 (x301-304)

Berlin - Prague : 1.75866 (x305-308)

Berlin - Zurich - Prague : 5.85154 (x309-312)

Berlin - Zurich - Milan - Prague : 7.59313 (x313-316)

Berlin - Brussels - Luxembourg - Milan - Prague : 10.4008 (x317-320)

Berlin - Zurich : 3.79854 (x321-324)

Berlin - Prague - Zurich : 3.81166 (x325-328)

Berlin - Prague - Milan - Zurich : 5.55325 (x329-332)

Berlin - Brussels - Luxembourg - Milan - Zurich : 8.50713 (x333-336)

Berlin - Brussels - Luxembourg : 4.79908 (x337-340)

Berlin - Amsterdam - Luxembourg : 5.03347 (x341-344)

Berlin - Amsterdam - Brussels - Luxembourg : 5.4544 (x345-348)

Berlin - Brussels - Amsterdam - Luxembourg : 6.73306 (x349-352)

Milan - Luxembourg - Brussels : 3.53988 (x353-356)

Milan - Paris - Brussels : 5.28062 (x357-360)

Milan - Luxembourg - Amsterdam - Brussels : 5.47386 (x361-364)

Milan - Zurich - Paris - Brussels : 5.76768 (x365-368)

Milan - Luxembourg - Amsterdam : 4.2964 (x369-372)

Milan - Luxembourg - Brussels - Amsterdam : 4.71733 (x373-376)

Milan - Paris - Brussels - Amsterdam : 6.45808 (x377-380)

Milan - Zurich - Paris - Brussels - Amsterdam : 6.94514 (x381-384)

Milan - Prague : 2.84412 (x385-388)

Milan - Zurich - Prague : 3.00347 (x389-392)

Milan - Zurich - Berlin - Prague : 6.50768 (x393-396)

Milan - Paris - Zurich - Prague : 8.99026 (x397-400)

Milan - Zurich : 0.950473 (x401-404)

Milan - Prague - Zurich : 4.89712 (x405-408)

Milan - Paris - Zurich : 6.93726 (x409-412)

Milan - Luxembourg - Brussels - Paris - Zurich : 8.35709 (x413-416)

Milan - Luxembourg : 2.75757 (x417-420)

Milan - Paris - Brussels - Luxembourg : 6.06293 (x421-424)

Milan - Zurich - Paris - Brussels - Luxembourg : 6.54999 (x425-428)

Milan - Paris - Brussels - Amsterdam - Luxembourg : 7.99691 (x429-432)

Brussels - Amsterdam : 1.17746 (x433-436)

Brussels - Luxembourg - Amsterdam : 2.32114 (x437-440)

Brussels - London - Amsterdam : 3.4356 (x441-444)

Brussels - Paris - London - Amsterdam : 5.2243 (x445-448)

Brussels - Berlin - Prague : 5.77544 (x449-452)

Brussels - Luxembourg - Milan - Prague : 6.38399 (x453-456)

Brussels - Amsterdam - Berlin - Prague : 6.43076 (x457-460)

Brussels - Luxembourg - Milan - Zurich - Prague : 6.54335 (x461-464)

Brussels - Luxembourg - Milan - Zurich : 4.49035 (x465-468)

Brussels - Paris - Zurich : 4.81721 (x469-472)

Brussels - Paris - Milan - Zurich : 6.23109 (x473-476)

Brussels - Amsterdam - Luxembourg - Milan - Zurich : 6.42433 (x477-480)

Brussels - Luxembourg : 0.782305 (x481-484)

Brussels - Amsterdam - Luxembourg : 2.71629 (x485-488)

Brussels - London - Amsterdam - Luxembourg : 4.97443 (x489-492)

Brussels - Paris - London - Amsterdam - Luxembourg : 6.76313 (x493-496)

Amsterdam - Berlin - Prague : 5.2533 (x497-500)

Amsterdam - Brussels - Berlin - Prague : 6.9529 (x501-504)

Amsterdam - Luxembourg - Milan - Prague : 7.14052 (x505-508)

Amsterdam - Luxembourg - Milan - Zurich - Prague : 7.29987 (x509-512)
Amsterdam - Luxembourg - Milan - Zurich : 5.24688 (x513-516)
Amsterdam - Brussels - Luxembourg - Milan - Zurich : 5.6678 (x517-520)
Amsterdam - Brussels - Paris - Zurich : 5.99467 (x521-524)
Amsterdam - London - Paris - Zurich : 6.88094 (x525-528)
Amsterdam - Luxembourg : 1.53883 (x529-532)
Amsterdam - Brussels - Luxembourg : 1.95976 (x533-536)
Amsterdam - London - Brussels - Luxembourg : 4.2179 (x537-540)
Amsterdam - London - Paris - Brussels - Luxembourg : 6.0066 (x541-544)
Prague - Zurich : 2.053 (x545-548)
Prague - Milan - Zurich : 3.79459 (x549-552)
Prague - Berlin - Zurich : 5.5572 (x553-556)
Prague - Milan - Paris - Zurich : 9.78138 (x557-560)
Prague - Milan - Luxembourg : 5.60169 (x561-564)
Prague - Zurich - Milan - Luxembourg : 5.76104 (x565-568)
Prague - Berlin - Brussels - Luxembourg : 6.55774 (x569-572)
Prague - Berlin - Amsterdam - Luxembourg : 6.79213 (x573-576)
Zurich - Milan - Luxembourg : 3.70804 (x577-580)
Zurich - Paris - Brussels - Luxembourg : 5.59952 (x581-584)
Zurich - Milan - Paris - Brussels - Luxembourg : 7.0134 (x585-588)
Zurich - Paris - Brussels - Amsterdam - Luxembourg : 7.5335 (x589-592)

D.2. More Efficient Use of Network Links

An improvement which reduces the number of network channels further has been achieved : assuming that WDM/OTDM is applied, it has been tried to make more efficient use of the channels (fill, if possible, all the available channel bandwidth) in order to reduce the overall number of channels. This has been done by adding into the objective function of the LP problem the variables representing the wavelengths capacities, multiplied with some *weighting factors*. The factors had smaller values for lower index wavelengths (λ_0, λ_1, ...) and larger values for high index wavelengths (..., λ_{31}, λ_{32}). In particular, the factor assigned to λ_0 is equal to 10 and the preceding factors are increased by 10 for each wavelength. Thus, λ_1 will be multiplied by 20, λ_2 by 30 and λ_{31} by 320. In that way, the LP solver will be forced to use the lower index channels, in order to reduce the overall cost (value of the objective function). Indeed, the output of lp_solve_2.0 shows, for this alternative LP model, that low index channels are *highly utilised*, whereas many of the high index channels have *zero capacity*.

For example, while, with conventional approach there were 31 paths active over the link Paris-London, with 31 channels being utilised, and λ_{15} not being used, in the alternative approach (using factors) there is the example of link Milan - Prague, over which there are 30 paths sharing only 15 channels (some of the channels are shared between 3 or 4 s-d pairs). The utilisation of wavelengths for these two cases, is presented in next section.

As an overall, this alternative approach gives as output *496* wavelength variables with non-zero value, which, however, although more than the 383 non-zero values of the previous case, *make use of only 335 channels* (wavelengths). This is *significant progress*, compared to the 348 channels, previously used.

The distribution of channels on the 17 established links of the network, for the two cases, assuming use of WDM/OTDM technology, is depicted in fig. 4.4.

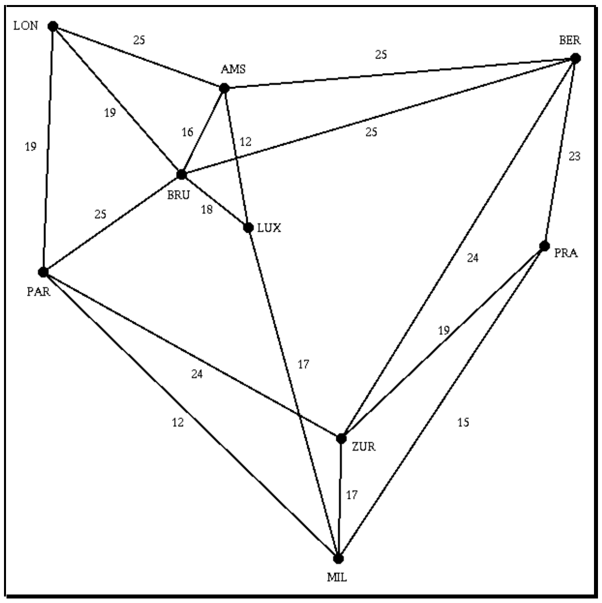


Fig. D.2.4 : Distribution of channels in the 9 central nodes of EON (a) Conventional approach, (b) Weighting Factors approach

D.3. Utilisation of Wavelengths for the Conventional and Weighting Factors Approaches

Conventional Approach :

Paris - London : λ_0:1, λ_1:1, λ_2:1, λ_3:1, λ_4:1, λ_5:1, λ_6:1, λ_7:1, λ_8:1, λ_9:1, λ_{10}:1, λ_{11}:1, λ_{12}:1, λ_{13}:1, λ_{14}:1, λ_{15}:0, λ_{16}:1, λ_{17}:1, λ_{18}:1, λ_{19}:1, λ_{20}:1, λ_{21}:1, λ_{22}:1, λ_{23}:1, λ_{24}:1, λ_{25}:1, λ_{26}:1, λ_{27}:1, λ_{28}:1, λ_{29}:1, λ_{30}:1, λ_{31}:1

Use of Weighting Factors :

Milan - Prague : λ_0:3, λ_1:2, λ_2:2, λ_3:2, λ_4:3, λ_5:2, λ_6:2, λ_7:2, λ_8:4, λ_9:1, λ_{10}:2, λ_{11}:2, λ_{12}:0, λ_{13}:0, λ_{14}:1, λ_{15}:0, λ_{16}:0, λ_{17}:0, λ_{18}:1, λ_{19}:0, λ_{20}:1, λ_{21}:0, λ_{22}:0, λ_{23}:0, λ_{24}:0, λ_{25}:0, λ_{26}:0, λ_{27}:0, λ_{28}:0, λ_{29}:0, λ_{30}:0, λ_{31}:0

APPENDIX E

CODES

E.1. Virtual Topology Design Code

```
/*******************************************************************************
Program: eon.cc

Author: D. Gavalas, E.S.E., MSc in Telecommunications and Information Systems

Date: 04/04/97

This program creates a file appropriate for input of "lp_solve_2.0" : an objective function, as well as a
number of constraints related to the "network optimisation by minimum cost" problem, are produced and
written into this file : "lp_input.txt". After the creation of that file, we can run the "lp_solve_2.0"
program, as follows :
                                        lp_solve < lp_input.txt
The program takes as input the number of nodes consisting the network and a file containing information
about the names of nodes, their location (coordinates) and the traffic corresponding to each potential
node pair. Thus, the program is called in the following way :
                                             eon <filename>
where <filename> is the name of the program's input file, described above.

It should be noted that the routing algorithm implemented, allows only direct, two-hop and three-hop
paths to be used.

Apart from the creation of the lp_solve input file, the program prints in the standard output the Nodes'
names, their coordinates, the Traffic Matrix, the table with the links' labels, as well as all the available
network's paths and their lengths.

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

#include <iostream.h>
#include <fstream.h>
#include <math.h>
#define km_per_unit 200   // The unit of distance is 200 km

int num_of_nodes;
void Read_File (ifstream&, char [][11], float [][2], float **);
void Sort (int [], int);

main(int argc, char *argv[])
{

  if (argc != 2) {                    // Wrong usage
    cout << "Usage : eon <filename> \n";
    return 1;
  }

  ifstream in (argv[1], ios::in);

  if (!in) { // Unable to open the file
    cout << "Cannot open " << argv[1] << " for input." << endl;
    exit(1);
```

```
}

int i,j,k,l,num_of_hops;

cout << "Insert the number of Network's nodes : ";
cin >> num_of_nodes;

// Evaluate the number of links in the Network
int num_of_links = num_of_nodes * (num_of_nodes-1) / 2;

cout << "Insert the maximum number of hops of the routing algorithm used : ";
cin >> num_of_hops;

/* Evaluate the number of direct, two-hops and three hops paths corresponding
   to a specific pair of source-destination node */
int num_of_paths_per_pair;
if (num_of_hops == 1)
  num_of_paths_per_pair = 1;
else if (num_of_hops == 2)
  num_of_paths_per_pair = num_of_nodes - 1;
else if (num_of_hops == 3)
  num_of_paths_per_pair = (num_of_nodes-1) +
    (num_of_nodes-2) * (num_of_nodes-3);
else {
  cout << "The program cannot support " << num_of_hops
    << "-hops routing algorithm." << endl;
  return(1);
}

// Evaluate the number of direct, two-hop and 3-hop paths in the Network
int num_of_paths = num_of_links * num_of_paths_per_pair;

float *traffic_matrix[num_of_nodes];
for (i = 0; i < num_of_nodes; i++)  /* Memory allocation for the input table
                                       of coefficients */
  traffic_matrix[i] = new float[num_of_nodes];

char Nodes[num_of_nodes][11];

float coordinates[num_of_nodes][2];

Read_File (in, Nodes, coordinates, traffic_matrix);

float distance[num_of_nodes][num_of_nodes];
for (i = 0; i < num_of_nodes; i++)     // Calculate the distance between 2 nodes
  for (j = 0; j < num_of_nodes; j++)
    distance[i][j] = sqrt(pow(coordinates[i][0]-coordinates[j][0],2) +
                          pow(coordinates[i][1]-coordinates[j][1],2));

ofstream out("lp_input.txt");
if (!out) {
  cout << "Cannot open file.";
  return 1;
}

cout << endl << "Traffic Matrix :" << endl;
for (i = 0; i < num_of_nodes; i++)          // Print traffic matrix
  for (j = 0; j < num_of_nodes; j++) {
    cout << traffic_matrix[i][j] << " ";
    if (j == num_of_nodes-1)
    cout << endl;
  }

// Create a Links' Label Matrix
int labels[num_of_nodes][num_of_nodes];

int label = 0;
for (i = 0; i < num_of_nodes-1; i++)
  for (j = i+1; j < num_of_nodes; j++) {
    labels[i][j] = labels[j][i] = label;
    label++;
  }

for (i=0; i < num_of_nodes; i++)
  labels[i][i] = -1;

cout << endl << "Network's Links Labels :" << endl;
for (i = 0; i < num_of_nodes; i++)          // Print the labels' matrix
  for (j = 0; j < num_of_nodes; j++) {
    cout << labels[i][j] << " ";
    if (j == num_of_nodes-1)
    cout << endl;
  }

cout << endl << "Number of paths : " << num_of_paths << endl;

int path_matrix[num_of_paths][num_of_hops + 1];   /* This array includes all the
                                                    available network's paths,
                                                    as well as their total length */
float path_length[num_of_paths];    // The total length of each path

int path_number, count;

if (num_of_hops == 3) {
  path_number = 0;  // The path's serial number
  for (i = 0; i < num_of_nodes-1; i++)    // Construct the path matrix
    for (j = i+1; j < num_of_nodes; j++) {
    path_matrix[path_number][0] = path_matrix[path_number][1] =
      path_matrix[path_number][2] = i;            // Direct path
    path_matrix[path_number][3] = j;
    path_length[path_number] =
      distance[path_matrix[path_number][0]][path_matrix[path_number][3]];
    path_number++;

    k = 0;
    while (k < num_of_nodes) {          // Two-hop path
      path_matrix[path_number][0] = i;
      path_matrix[path_number][3] = j;
      while (k == i || k == j)
        k++;
      if (k < num_of_nodes) {
        path_matrix[path_number][1] = path_matrix[path_number][2] = k;
        path_length[path_number] =
          distance[path_matrix[path_number][0]][path_matrix[path_number][1]] +
                distance[path_matrix[path_number][1]][path_matrix[path_number][3]];
        path_number++;
        l = 0;
        while (l < num_of_nodes) {        // Two-hop path
```

```
            path_matrix[path_number][0] = i;
            path_matrix[path_number][1] = k;
            path_matrix[path_number][3] = j;
            while (l == i || l == j || l == k)
                    l++;
            if (l < num_of_nodes) {
                    path_matrix[path_number][2] = l;
                    path_length[path_number] =
                      distance[path_matrix[path_number][0]][path_matrix[path_number][1]] +
                        distance[path_matrix[path_number][1]][path_matrix[path_number][2]] +
                          distance[path_matrix[path_number][2]][path_matrix[path_number][3]];
                    path_number++;
            }
            l++;
          }
          k++;
        }
      }
      }
  }
  else if (num_of_hops == 2) {
    path_number = 0;  // The path's serial number
    for (i = 0; i < num_of_nodes-1; i++)    // Construct the path matrix
      for (j = i+1; j < num_of_nodes; j++) {
     path_matrix[path_number][0] = path_matrix[path_number][1] = i;      // Direct path
     path_matrix[path_number][2] = j;
     path_length[path_number] =
       distance[path_matrix[path_number][0]][path_matrix[path_number][2]];
     path_number++;
       k = 0;
     while (k < num_of_nodes) {           // Two-hop path
       path_matrix[path_number][0] = i;
       path_matrix[path_number][2] = j;
       while (k == i || k == j)
         k++;
       if (k < num_of_nodes) {
         path_matrix[path_number][1] = k;
         path_length[path_number] =
           distance[path_matrix[path_number][0]][path_matrix[path_number][1]] +
                   distance[path_matrix[path_number][1]][path_matrix[path_number][2]];
         path_number++;
         k++;
       }
      }
      }
  }
  else if (num_of_hops == 1) {
    path_number = 0;  // The path's serial number
    for (i = 0; i < num_of_nodes-1; i++)    // Construct the path matrix
      for (j = i+1; j < num_of_nodes; j++) {
     path_matrix[path_number][0] = i;      // Direct path
     path_matrix[path_number][1] = j;
     path_length[path_number] = distance[i][j];
     path_number++;
      }
  }

  cout << endl;
if (num_of_hops == 3) {
    for (i = 0; i < num_of_paths; i++) {   /* Print all the networks paths,
                                              and the corresponding total length */
      cout << "x" << i+num_of_links << " = ";
      if ((path_matrix[i][0] == path_matrix[i][1]) && (path_matrix[i][1] == path_matrix[i][2])) {
      cout << Nodes[path_matrix[i][0]] << " - ";
      cout <<  Nodes[path_matrix[i][3]] << " - ";
      cout << "Traffic::" << traffic_matrix[path_matrix[i][0]][path_matrix[i][3]] +
        traffic_matrix[path_matrix[i][3]][path_matrix[i][0]];
      cout << " Length:" << path_length[i];
      cout << "  (x" << labels[path_matrix[i][0]][path_matrix[i][3]] << ")";
       }
      else if (path_matrix[i][1] == path_matrix[i][2]) {
      cout << Nodes[path_matrix[i][0]] << " - ";
      cout << Nodes[path_matrix[i][2]] << " - ";
      cout << Nodes[path_matrix[i][3]] << " - ";
      cout << path_length[i];
      cout << "  (x" << labels[path_matrix[i][0]][path_matrix[i][3]] << ")";
       }
      else {
      for (j = 0; j < 4; j++)
        cout << Nodes[path_matrix[i][j]] << " - ";
      cout << path_length[i];
      cout << "  (x" << labels[path_matrix[i][0]][path_matrix[i][3]] << ")";
       }
      cout << endl;
    }
  }
  else if (num_of_hops == 2) {
    for (i = 0; i < num_of_paths; i++) {   /* Print all the networks paths,
                                              and the corresponding total length */
      cout << "x" << i+num_of_links << " = ";
      if (i % (num_of_nodes-1) != 0) {
      for (j = 0; j < 3; j++)
        cout << Nodes[path_matrix[i][j]] << " - ";
      cout << path_length[i];
      cout << "  (x" << labels[path_matrix[i][0]][path_matrix[i][2]] << ")";
       }
      else {
      cout << Nodes[path_matrix[i][0]] << " - ";
      cout << Nodes[path_matrix[i][2]] << " - ";
      cout << path_length[i];
      cout << "  (x" << labels[path_matrix[i][0]][path_matrix[i][2]] << ")";
       }
      cout << endl;
    }
  }

// Insert the objective function into the lp_solve input file

  out << "min: ";
  for (i = 0; i < num_of_paths-num_of_paths_per_pair; i+=num_of_paths_per_pair) { /* This
includes the cost of the nodes
                                                                  (the cost of each node is equal to
                                                                  the sum of the costs of the links
                                                                  attached to that node) and the cost
                                                                  of links (which is equal to length
of link multiplied with its capacity */
    out << 2 + (km_per_unit * path_length[i]) << " x" << i/num_of_paths_per_pair << " + ";
  }
```

```
    out << 2 + (km_per_unit * path_length[num_of_paths - num_of_paths_per_pair]) << " x" <<
(num_of_paths - 1) / num_of_paths_per_pair << ";" << endl << endl;

/*  for (i = 0; i < num_of_paths-1; i++)    // This includes the cost of paths
    out << km_per_unit * path_length[i] << " x" << i+num_of_links << " + ";

  out << km_per_unit * path_length[num_of_paths-1] << " x" << num_of_paths+num_of_links-1
    << ";" << endl << endl;
*/

  /* Constraint in cuts around nodes (the links attached to each node are
     supposed to have total capacity bigger or equal to the sum of capacities,
     row & column, of the traffic matrix) */

  float sum = 0.0;    // Sum of row & column capacities
  for (i = 0; i < num_of_nodes; i++) {
    for (j = 0; j < num_of_nodes; j++) {
      sum += traffic_matrix[i][j] + traffic_matrix[j][i];
      if (labels[i][j] >= 0)
      if (j < num_of_nodes-2)
        out << "x" << labels[i][j] << " + ";
      else if (j == num_of_nodes-2 && labels[i][j+1] >= 0)
        out << "x" << labels[i][j] << " + ";
      else
        out << "x" << labels[i][j] << " >= " << sum << ";" << endl;
    }
    sum = 0;
  }

  /* An another constraint is added if we remove each one of the links between 2 nodes
     and then require the rest of the links attached to these 2 nodes to have total capacity
     equal to the sum of traffic units with destination the 2 nodes */

  int array_of_labels[num_of_nodes-2];
  int counter;
  out << endl;

  for (i = 0; i < num_of_links; i++) {
    for (j = 0; j < num_of_nodes; j++) { // find the coordinates of the link with label = i
      for (k = 0; k < num_of_nodes; k++)
      if (labels[j][k] == i)
        break;
      if (labels[j][k] == i)
      break;
    }
    sum = 0.0; /* Sum of traffic units with destination the 2 nodes that the
                 specific link connects, except the ones carried by that link */

    counter = 0; // Number of links attached to the 2 nodes except the one removed
    for (l = 0; l < num_of_nodes; l++) {
      sum += traffic_matrix[j][l] + traffic_matrix[l][k];
      if (labels[j][l] != -1 && labels[j][l] != i) {
      array_of_labels[counter] = labels[j][l];
      counter++;
      }
      if (labels[l][k] != -1 && labels[l][k] != i) {
      array_of_labels[counter] = labels[l][k];
      counter++;
      }
    }

    sum -= 2 * traffic_matrix[j][k];
    Sort(array_of_labels, counter);

    for (k = 0; k < counter; k++)
      if (k < counter-1)
      out << "x" << array_of_labels[k] << " + ";
      else
      out << "x" << array_of_labels[k] << " >= " << 2*sum << ";" << endl;
  } // for

  out << endl;

  for (i = 0; i < num_of_links; i++)
    out << " int x" << i << ";" << endl;
  out << endl;

  /* At this point we introduce into the LP problem some extra variables, each
     one of which represents the amount of traffic carried out by a stream
     (corresponding to a specific pair of source-destination nodes). A constraint is
     introduced, if we require the sum of the traffics carried by those streams, to
     be equal to the traffic between the 2 nodes, given by the traffic matrix. */
  if (num_of_hops > 1)
    for (i = 0; i < num_of_links; i++) {
      k = i * num_of_paths_per_pair;
      sum = traffic_matrix[path_matrix[k][0]][path_matrix[k][num_of_hops]] +
      traffic_matrix[path_matrix[k][num_of_hops]][path_matrix[k][0]];
      for (j = k; j < (i+1)*num_of_paths_per_pair; j++)
      if (j != (i+1)*num_of_paths_per_pair-1)
        out << "x" << j+num_of_links << " + ";
      else
        out << "x" << j+num_of_links << " = " << sum << ";" << endl;
    }

  out << endl;

  /* An extra constraint is added by requiring the sum of streams' traffic, for
     streams that use a specific link, to be equal to the capacity of that link. */

  for (i = 0; i < num_of_links; i++) {
    for (j = 0; j < num_of_nodes; j++) {  // Find the coordinates (j,k) in the label
      for (k = 0; k < num_of_nodes; k++)  // matrix of the link with label = i
      if (labels[j][k] == i)
        break;
      if (labels[j][k] == i)
      break;
    }

    if (num_of_hops == 3) {
      count = 0;
      for (l = 0; l < num_of_paths; l++)   /* Calculate how many streams use
                                              the current link */
      if ((path_matrix[l][0] == j && path_matrix[l][1] == k) ||
          (path_matrix[l][1] == j && path_matrix[l][2] == k) ||
          (path_matrix[l][2] == j && path_matrix[l][3] == k) ||
          (path_matrix[l][0] == k && path_matrix[l][1] == j) ||
          (path_matrix[l][1] == k && path_matrix[l][2] == j) ||
          (path_matrix[l][2] == k && path_matrix[l][3] == j))
        count++;

      counter = 0;
      for (l = 0; l < num_of_paths; l++) {
      if ((path_matrix[l][0] == j && path_matrix[l][1] == k) ||
          (path_matrix[l][1] == j && path_matrix[l][2] == k) ||
          (path_matrix[l][2] == j && path_matrix[l][3] == k) ||
          (path_matrix[l][0] == k && path_matrix[l][1] == j) ||
          (path_matrix[l][1] == k && path_matrix[l][2] == j) ||
          (path_matrix[l][2] == k && path_matrix[l][3] == j)) {
        if (counter < count-1)
          out << "x" << l+num_of_links << " + ";
        else
          out <<  "x" << l+num_of_links << " = x" << i << ";" << endl;
        counter++;
      }
      }
    }
    else if (num_of_hops == 2) {
      count = 0;
      for (l = 0; l < num_of_paths; l++)   /* Calculate how many streams use
                                              the current link */
      if ((path_matrix[l][0] == j && path_matrix[l][1] == k) ||
          (path_matrix[l][1] == j && path_matrix[l][2] == k) ||
          (path_matrix[l][0] == k && path_matrix[l][1] == j) ||
          (path_matrix[l][1] == k && path_matrix[l][2] == j))
        count++;

      counter = 0;
      for (l = 0; l < num_of_paths; l++) {
      if ((path_matrix[l][0] == j && path_matrix[l][1] == k) ||
          (path_matrix[l][1] == j && path_matrix[l][2] == k) ||
          (path_matrix[l][0] == k && path_matrix[l][1] == j) ||
          (path_matrix[l][1] == k && path_matrix[l][2] == j)) {
        if (counter < count-1)
          out << "x" << l+num_of_links << " + ";
        else
          out <<  "x" << l+num_of_links << " = x" << i << ";" << endl;
        counter++;
      }
      }
    }
  }
  out << endl;


} // Main



/* This function is used for reading the input file containing information about
   the names of nodes, their location (coordinates) and the traffic corresponding
   to each potential node pair */
void Read_File (ifstream& in, char Nodes[][11],
                  float coordinates[][2], float **traffic_matrix)
{
  int i,j,x,y;
  char ch;
  for (i = 0; i < num_of_nodes; i++) { /* Create the array of Nodes' names
                                          and coordinates */
    in >> Nodes[i];         // Name of node
    in >> coordinates[i][0];   // x cordinate
    in >> coordinates[i][1];   // y coordinate
  }
//  in >> ch;

  cout << endl << "Coordinates :" << endl;
  for (i = 0; i<num_of_nodes; i++)
    cout << Nodes[i] << " : x=" << coordinates[i][0] << ", y=" << coordinates[i][1] << endl;

  for (i = 0; i < num_of_nodes; i++) {   // Reading information about the traffic
    for (j = 0; j < num_of_nodes; j++) {
      in >> x;
      in >> y;
      in >> traffic_matrix[i][j];
    }
  }
  in.close();

}  // Read_File



/* This function sorts an array of "count" numbers in increasing order,
   using the Selection-Sort algorithm */
void Sort (int numbers[], int count)
{
  int i,j,t;

  for (i = 1; i < count; i++) {
    t = numbers[i];
    j = i;
    while (numbers[j-1] > t) {
      numbers[j] = numbers[j-1];
      j--;
    }
    numbers[j] = t;
  }

} // Sort
```

E.2. Routing & Wavelength Assignment Code

/**
Program: wavelength.cc

Author: D. Gavalas, E.S.E., MSc in Telecommunications and Information Systems

Date: 04/04/97

This program creates a file appropriate for input of "lp_solve_2.0" (an objective function, as well as a number of constraints related to the "wavelength assignment optimisation by minimum cost" problem are produced and written into this file : "wavel.txt"). After the creation of that file, we can run the "lp_solve_2.0" program, as follows :

lp_solve < wavel.txt

The program takes as input the number of nodes consisting the network and a file containing information about the names of nodes, their location (coordinates) and the traffic corresponding to each potential node pair. Thus, the program is called in the following way :

wavelength <filename>

where <filename> is the name of the program's input file, described above.

Apart from the creation of the lp_solve input file, the program prints in the standard output the Nodes' names, the existing connections, the Traffic Matrix, the Adjacency Matrix, as well as all the available network's paths and their lengths, followed by a list of the 4 shortest paths for each of the node pairs.

```
******************************************************************************/

  #include <iostream.h>
  #include <fstream.h>
  #include <math.h>

  #define km_per_unit 200   // The unit of distance is 200 km
  #define NUM_OF_SHORTEST_PATHS 4  /* The program selects the 3 shortests paths for each
                                    node pair, in order to reduce the number of variables */
  #define MAX_NUM_OF_PATHS 30  /* We assume that there is not possibility of having more
                                    than 30 direct, 2, 3 and 4-hop paths for any node pair */
  #define NUM_OF_HOPS 4         // The maximum possible number of hops we consider
  #define WAVELENGTH_CAPACITY 2.5   // The capacity that a channel (wavelength) can carry

  int num_of_nodes,num_of_wavelengths;
  void Read_File (ifstream&, char [][11], float [][2], float **, int **, int&);
  void Sort (int [], int);
  void Sort_Paths (int *[], float [], int);
  void Initialize (int *[], float []);

  main(int argc, char *argv[])
  {

   if (argc != 2) {                   // Wrong usage of the program
     cout << "Usage : eon <filename> \n";
     return 1;
   }

   ifstream in (argv[1], ios::in);

   if (!in) { // Unable to open the file
     cout << "Cannot open " << argv[1] << " for input." << endl;
     exit(1);
   }
   int i,j,k,l,m,num_of_links;

   cout << "Insert the number of Network's nodes : ";
   cin >> num_of_nodes;

   cout << endl << "Insert the number of wavelengths in the network : ";
   cin >> num_of_wavelengths;

  /* Evaluate the number of direct, two-hops, three-hops and four-hops paths corresponding
     to a specific pair of source-destination node */
   int num_of_paths_per_pair = 1 + (num_of_nodes-2) + (num_of_nodes-2) * (num_of_nodes-3) +
      (num_of_nodes-2) * (num_of_nodes-3) * (num_of_nodes - 4);
```

```
float *traffic_matrix[num_of_nodes];
for (i = 0; i < num_of_nodes; i++)  // Memory allocation for the Traffic Matrix
  traffic_matrix[i] = new float[num_of_nodes];

char Nodes[num_of_nodes][11];

float coordinates[num_of_nodes][2];

int *AdjacencyMatrix[num_of_nodes];
for (i = 0; i < num_of_nodes; i++)   // Memory allocation for the Adjacency Matrix
  AdjacencyMatrix[i] = new int[num_of_nodes];

int labels[num_of_nodes][num_of_nodes];  // The matrix that contains the links labels

for (i = 0; i < num_of_nodes; i++)   // Initialization of tables
  for (j = 0; j < num_of_nodes; j++) {
    traffic_matrix[i][j] = 0.0;
    AdjacencyMatrix[i][j] = 0;
    labels[i][j] = -1;
  }

Read_File (in, Nodes, coordinates, traffic_matrix, AdjacencyMatrix, num_of_links);

int NodePairs[num_of_links][2];  /* An array that stores the source and
                                    destination node of each existing link */
int NodesConnections [num_of_nodes];  /* The number of connections (number
                                          of attached links) of each node */
for (i = 0; i < num_of_nodes; i++)
  NodesConnections[i] = 0;


k = 0;
for (i = 0; i < num_of_nodes; i++) {   // Construct the array of Node pairs
  for (j = i+1; j < num_of_nodes; j++)
    if (AdjacencyMatrix[i][j] == 1) {
        NodePairs[k][0] = i;
        NodePairs[k][1] = j;
        NodesConnections[i]++;
        NodesConnections[j]++;
        labels[i][j] = labels[j][i] = k++;
    }
  cout << endl;
}

for (i = 0; i < num_of_links; i++)
  cout << Nodes[NodePairs[i][0]] << " - " << Nodes[NodePairs[i][1]] << endl;

cout << endl << "NODES CONNECTIONS" << endl;
for (i = 0; i < num_of_nodes; i++)
  cout << Nodes[i] << " : " << NodesConnections[i] << endl;

// Evaluate the number of node pairs in the Network
int num_of_node_pairs = num_of_nodes * (num_of_nodes - 1) / 2;

/* Evaluate the maximum possible number of direct, two-hop and
   3-hop paths in the Network */
int num_of_paths = num_of_node_pairs * num_of_paths_per_pair;
```

```
  float distance[num_of_nodes][num_of_nodes];
  for (i = 0; i < num_of_nodes; i++)         // Calculate the distance between 2 nodes
    for (j = 0; j < num_of_nodes; j++)
      distance[i][j] = sqrt(pow(coordinates[i][0]-coordinates[j][0],2) +
                                          pow(coordinates[i][1]-coordinates[j][1],2));

  ofstream out("factor.txt");
  if (!out) {
    cout << "Cannot open file.";
    return 1;
  }

  cout << endl << "Traffic Matrix :" << endl;
  for (i = 0; i < num_of_nodes; i++)          // Print traffic matrix
    for (j = 0; j < num_of_nodes; j++) {
      cout << traffic_matrix[i][j] << " ";
      if (j == num_of_nodes-1)
              cout << endl;
    }

  int int_part;
  float mod;
  for (i = 0; i < num_of_nodes; i++)        /* Round the elements of traffic matrix
                                              to the nearest multiple of 2.5 */
    for (j = 0; j < num_of_nodes; j++) {
      int_part = int(traffic_matrix[i][j] / 2.5);
      mod = traffic_matrix[i][j] - 2.5 * int_part;
if (traffic_matrix[i][j] != 0)
              traffic_matrix[i][j] = 2.5 * (int_part + 1);
}


  cout << endl << "Truncated Traffic Matrix :" << endl;
  for (i = 0; i < num_of_nodes; i++)          // Print traffic matrix
    for (j = 0; j < num_of_nodes; j++) {
      cout << traffic_matrix[i][j] << " ";
      if (j == num_of_nodes-1)
              cout << endl;
    }



  cout << endl << "Network's Links Labels :" << endl;
  for (i = 0; i < num_of_nodes; i++)          // Print the labels' matrix
    for (j = 0; j < num_of_nodes; j++) {
      if (labels[i][j] >= 0)
              cout << labels[i][j] << " ";
      else
              cout << " - ";
      if (j == num_of_nodes-1)
              cout << endl;
    }

//  cout << endl << "Number of paths : " << num_of_paths << endl;

  int path_matrix[num_of_paths][NUM_OF_HOPS + 1];   /* This array includes all the
                                                       available network's paths,
                                                       as well as their total length */
  float path_length[num_of_paths];    // The total length of each path
```

```
int path_number;
int count = 0;

path_number = 0;  // The path's serial number
for (i = 0; i < num_of_nodes-1; i++)    // Construct the path matrix
  for (j = i+1; j < num_of_nodes; j++) {
    if (AdjacencyMatrix[i][j] == 1) {
            path_matrix[path_number][0] = path_matrix[path_number][1] =
                path_matrix[path_number][2] = path_matrix[path_number][3] = i;  // Direct path
            path_matrix[path_number][4] = j;
            path_length[path_number] =
              distance[path_matrix[path_number][0]][path_matrix[path_number][4]];
            count++;
            }
    else {
            path_matrix[path_number][0] = path_matrix[path_number][1] =
              path_matrix[path_number][2] = path_matrix[path_number][3] =
                path_matrix[path_number][4] = -1;
              path_length[path_number] = 0;
    }
    path_number++;

    k = 0;
    while (k < num_of_nodes) {           // Two-hop path
            path_matrix[path_number][0] = i;
              path_matrix[path_number][4] = j;
            while (k == i || k == j)
              k++;
            if (k < num_of_nodes) {
              if (AdjacencyMatrix[i][k] == 1 && AdjacencyMatrix[k][j] == 1) {
                path_matrix[path_number][1] = path_matrix[path_number][2] =
                  path_matrix[path_number][3] = k;
                path_length[path_number] =
                  distance[path_matrix[path_number][0]][path_matrix[path_number][1]] +
                          distance[path_matrix[path_number][1]][path_matrix[path_number][4]];
                  count;
              }
              else {
                path_matrix[path_number][0] = path_matrix[path_number][1] =
                          path_matrix[path_number][2] = path_matrix[path_number][3] =
                            path_matrix[path_number][4] = -1;
                path_length[path_number] = 0;
              }
              path_number++;

              l = 0;
              while (l < num_of_nodes) {        // Three-hop path
                path_matrix[path_number][0] = i;
                path_matrix[path_number][1] = k;
                path_matrix[path_number][4] = j;
                  while (l == i || l == j || l == k)
                          l++;
                if (l < num_of_nodes) {
                  if (AdjacencyMatrix[i][k] == 1 && AdjacencyMatrix[k][l] == 1 &&
                              AdjacencyMatrix[l][j] == 1) {
                          path_matrix[path_number][2] = path_matrix[path_number][3] = l;
                          path_length[path_number] =
                            distance[path_matrix[path_number][0]][path_matrix[path_number][1]] +
                    distance[path_matrix[path_number][1]][path_matrix[path_number][2]] +
                      distance[path_matrix[path_number][2]][path_matrix[path_number][4]];
                  count++;
              }
              else {
                  path_matrix[path_number][0] = path_matrix[path_number][1] =
                    path_matrix[path_number][2] = path_matrix[path_number][3] =
                      path_matrix[path_number][4] = -1;
                  path_length[path_number] = 0;
                  }
              path_number++;

              m = 0;
              while (m < num_of_nodes) {          // Four-hop
                  path_matrix[path_number][0] = i;
                  path_matrix[path_number][1] = k;
                  path_matrix[path_number][2] = l;
                  path_matrix[path_number][4] = j;
                  while (m == i || m == j || m == k || m == l)
                    m++;
                  if (m < num_of_nodes) {
                    if (AdjacencyMatrix[i][k] == 1 && AdjacencyMatrix[k][l] == 1 &&
                        AdjacencyMatrix[l][m] == 1 && AdjacencyMatrix[m][j] == 1) {
                      path_matrix[path_number][3] = m;
                      path_length[path_number] =
                        distance[path_matrix[path_number][0]][path_matrix[path_number][1]] +
                          distance[path_matrix[path_number][1]][path_matrix[path_number][2]] +
                           distance[path_matrix[path_number][2]][path_matrix[path_number][3]] +
                              distance[path_matrix[path_number][3]][path_matrix[path_number][4]];
                      count++;
                    }
                    else {
                      path_matrix[path_number][0] = path_matrix[path_number][1] =
                        path_matrix[path_number][2] = path_matrix[path_number][3] =
                                path_matrix[path_number][4] = -1;
                      path_length[path_number] = 0;
                    }
                    path_number++;
                  }
                  m++;
              }
            }
            l++;
          }
        }
        k++;
    }
  }


  cout << endl;
  int counter = 0;
  int ShortestPaths[NUM_OF_SHORTEST_PATHS * num_of_node_pairs][NUM_OF_HOPS +
1];                 // These are the 3-shortest paths of every node pair in the network
  float ShortestLengths[NUM_OF_SHORTEST_PATHS * num_of_node_pairs];
  int *PathsArray[MAX_NUM_OF_PATHS];  // The paths of the current node pair
  for (i = 0; i < MAX_NUM_OF_PATHS; i++)   // Memory allocation
    PathsArray[i] = new int[NUM_OF_HOPS + 1];
```

```
   float LengthsArray[MAX_NUM_OF_PATHS];
   Initialize(PathsArray, LengthsArray);

   counter = 0;
   for (i = 0; i < num_of_paths; i++) {   /* Print all the networks paths,
                                             and the corresponding total length */
     if ((path_matrix[i][0] == path_matrix[i][1]) && (path_matrix[i][1] == path_matrix[i][2]) &&
            (path_matrix[i][2] == path_matrix[i][3]) && (path_matrix[i][0] != -1)) {
       cout << "x" << i+num_of_links << " = ";
       cout << Nodes[path_matrix[i][0]] << " - ";
       cout <<  Nodes[path_matrix[i][4]] << " - ";
       cout << "Traffic :" << traffic_matrix[path_matrix[i][0]][path_matrix[i][4]] +
            traffic_matrix[path_matrix[i][4]][path_matrix[i][0]];
       cout << "  Length :" << path_length[i];
       cout << "  (x" << labels[path_matrix[i][0]][path_matrix[i][4]] << ")" << endl;
       PathsArray[counter] = path_matrix[i];
       LengthsArray[counter] = path_length[i];
       counter++;
     }
     else if (path_matrix[i][1] == path_matrix[i][2] && path_matrix[i][2] == path_matrix[i][3]
               && path_matrix[i][0] != -1) {
       cout << "x" << i+num_of_links << " = ";
       cout << Nodes[path_matrix[i][0]] << " - ";
       cout << Nodes[path_matrix[i][2]] << " - ";
       cout << Nodes[path_matrix[i][4]] << " - ";
       cout << path_length[i];
       cout << "  (x" << labels[path_matrix[i][0]][path_matrix[i][4]] << ")" << endl;
       PathsArray[counter] = path_matrix[i];
       LengthsArray[counter] = path_length[i];
       counter++;
     }
     else if (path_matrix[i][2] == path_matrix[i][3] && path_matrix[i][0] != -1) {
       cout << "x" << i+num_of_links << " = ";
       for (j = 0; j < NUM_OF_HOPS + 1; j++)
            if (j != NUM_OF_HOPS - 1)
              cout << Nodes[path_matrix[i][j]] << " - ";
       cout << path_length[i];
       cout << "  (x" << labels[path_matrix[i][0]][path_matrix[i][NUM_OF_HOPS]] << ")" <<
  endl;
       PathsArray[counter] = path_matrix[i];
       LengthsArray[counter] = path_length[i];
       counter++;
     }
     else if (path_matrix[i][0] != -1) {
       cout << "x" << i+num_of_links << " = ";
       for (j = 0; j < NUM_OF_HOPS + 1; j++)
            cout << Nodes[path_matrix[i][j]] << " - ";
       cout << path_length[i];
       cout << "  (x" << labels[path_matrix[i][0]][path_matrix[i][NUM_OF_HOPS]] << ")" <<
  endl;
       PathsArray[counter] = path_matrix[i];
       LengthsArray[counter] = path_length[i];
       counter++;
     }

     if (counter != 0 && (i+1) % num_of_paths_per_pair == 0) {
       Sort_Paths(PathsArray, LengthsArray, counter);
       cout << "x" << i / num_of_paths_per_pair << " : "
            << counter << " paths" << endl;
```

```
      for (j = i/num_of_paths_per_pair * NUM_OF_SHORTEST_PATHS;
               j < ((i/num_of_paths_per_pair) + 1) * NUM_OF_SHORTEST_PATHS; j++) {
            for (k = 0; k < NUM_OF_HOPS+ 1; k++)
              ShortestPaths[j][k] = PathsArray[j - i/num_of_paths_per_pair *
NUM_OF_SHORTEST_PATHS][k];
            ShortestLengths[j] = LengthsArray[j - i/num_of_paths_per_pair *
NUM_OF_SHORTEST_PATHS];
      }
      Initialize(PathsArray, LengthsArray);
      counter = 0;
    }
  }

  cout << "SHORTESTS PATHS : " << endl;
  for (i = 0; i < NUM_OF_SHORTEST_PATHS * num_of_node_pairs; i++) {
    if (ShortestPaths[i][0] == ShortestPaths[i][1] && ShortestPaths[i][1] == ShortestPaths[i][2]
&&
            ShortestPaths[i][2] == ShortestPaths[i][3])
      cout << Nodes[ShortestPaths[i][0]] << " - " <<  Nodes[ShortestPaths[i][NUM_OF_HOPS]]
<< " : "
            << ShortestLengths[i] << " (x" << i*num_of_wavelengths + num_of_links << "-"
              << i*num_of_wavelengths + num_of_links + num_of_wavelengths - 1 << ")" << endl;
    else if (ShortestPaths[i][1] == ShortestPaths[i][2] && ShortestPaths[i][2] ==
ShortestPaths[i][3])
      cout << Nodes[ShortestPaths[i][0]] << " - " <<  Nodes[ShortestPaths[i][1]] << " - "
            << Nodes[ShortestPaths[i][NUM_OF_HOPS]] << " : " << ShortestLengths[i] << " (x"
              << i*num_of_wavelengths + num_of_links << "-"
                << i*num_of_wavelengths + num_of_links + num_of_wavelengths - 1 << ")" << endl;
    else if (ShortestPaths[i][2] == ShortestPaths[i][3])
      cout << Nodes[ShortestPaths[i][0]] << " - " <<  Nodes[ShortestPaths[i][1]] << " - "
            << Nodes[ShortestPaths[i][2]] << " - " << Nodes[ShortestPaths[i][NUM_OF_HOPS]]
              << " : " << ShortestLengths[i] << " (x" << i*num_of_wavelengths + num_of_links <<
                  "-"  << i*num_of_wavelengths + num_of_links + num_of_wavelengths - 1 << ")" <<
                    endl;
    else
      cout << Nodes[ShortestPaths[i][0]] << " - " <<  Nodes[ShortestPaths[i][1]] << " - "
            << Nodes[ShortestPaths[i][2]] << " - " <<  Nodes[ShortestPaths[i][3]] << " - "
              << Nodes[ShortestPaths[i][NUM_OF_HOPS]] << " : " << ShortestLengths[i] << " (x"
                << i*num_of_wavelengths + num_of_links << "-"
                  << i*num_of_wavelengths + num_of_links + num_of_wavelengths - 1 << ")" <<
endl;
    if ((i+1) % NUM_OF_SHORTEST_PATHS == 0)
      cout << endl;
  }



  // Insert the OBJECTIVE FUNCTION into the lp_solve input file
  out << "min: ";
  for (i = 0; i < num_of_links - 1; i++)
    out << "x" << i << " + ";
  out << "x" << num_of_links - 1 << ";" << endl << endl;

  /* Constraint in cuts around nodes (the links attached to each node are
     supposed to have total capacity bigger or equal to the sum of traffic with
     source and destination this node) */
  float sum = 0.0;    // Sum of row & column capacities
  for (i = 0; i < num_of_nodes; i++) {
    k = 0;
  for (j = 0; j < num_of_nodes; j++) {
    sum += traffic_matrix[i][j] + traffic_matrix[j][i];
    if (AdjacencyMatrix[i][j] == 1) {
        if (k < NodesConnections[i] - 1) {
          out << "x" << labels[i][j] << " + ";
          k++;
        }
        else
          out << "x" << labels[i][j] << " >= " << sum << ";" << endl;
    }
  }
  sum = 0.0;
}


/* An another constraint is added if we remove each one of the links between 2 nodes
   and then require the rest of the links attached to these 2 nodes to have total capacity
   equal to the sum of traffic units with source or destination the 2 nodes */
int array_of_labels[num_of_nodes-2];
out << endl;

for (i = 0; i < num_of_links; i++) {
  j = NodePairs[i][0];    // Find the two nodes to which the current link is attached to
  k = NodePairs[i][1];
  sum = 0.0;    /* Sum of traffic units with source and destination the 2 nodes that the
                   specific link connects, except the ones carried by that link */

  counter = NodesConnections[j] + NodesConnections[k] -2;   /* Number of links attached
                                                                to the 2 nodes except
                                                                the one removed */
  m = 0;
  for (l = 0; l < num_of_nodes; l++) {
    sum += traffic_matrix[j][l] + traffic_matrix[l][j]
        + traffic_matrix[l][k] + traffic_matrix[k][l];
    if (labels[j][l] != -1 && labels[j][l] != i) {
        array_of_labels[m] = labels[j][l];
        m++;
    }
    if (labels[l][k] != -1 && labels[l][k] != i) {
        array_of_labels[m] = labels[l][k];
        m++;
    }
  }
  sum -= (2 * traffic_matrix[j][k]) + (2 * traffic_matrix[k][j]);
  Sort(array_of_labels, counter);

  for (k = 0; k < counter; k++)
    if (k < counter-1)
        out << "x" << array_of_labels[k] << " + ";
    else
        out << "x" << array_of_labels[k] << " >= " << sum << ";" << endl;
} // for

out << endl;


/* At this point we introduce into the LP problem some extra variables, each
   one of which represents the ammount of traffic carried out by a stream
   (corresponding to one wavelength of an existing node pair). A constraint is
```

```
     introduced, if we require the sum of the traffics carried by those streams, to
     be equal to the traffic between the 2 nodes, given by the traffic matrix. */
int VarsPerNodePair = NUM_OF_SHORTEST_PATHS * num_of_wavelengths;
for (i = 0; i < num_of_node_pairs; i++)
   for (j = 0; j < VarsPerNodePair; j++) {

     counter = 0;       // Find the source and destination node of the current node pair
     for (k = 0; k < num_of_nodes; k++) {
          for (l = k+1; l < num_of_nodes; l++)
            if (counter == i)
              break;
            else if (l < num_of_nodes - 1)
              counter++;
          if (counter == i)
            break;
          else
            counter++;
     }

     sum = traffic_matrix[k][l] + traffic_matrix[l][k];   /* The traffic between the nodes
                                                              that the current link connects */
     for (j = 0; j < VarsPerNodePair - 1; j++)
          out << "x" << i * VarsPerNodePair + j + num_of_links << " + ";
     out <<  "x" << (i + 1) * VarsPerNodePair - 1 + num_of_links << " = " << sum << ";" <<
endl;
   }

  out << endl;



  /* An extra constraint is added by requiring the sum of streams, for streams
     that use a specific link, to be equal to the capacity of that link. */

  int ParticipatingStreams[num_of_links][num_of_node_pairs *
NUM_OF_SHORTEST_PATHS];
                                         /* A list of streams that
                                            traverse a specific link */
  for (i = 0; i < num_of_links; i++) {
    j = NodePairs[i][0];    // Find the two nodes to which the current link is attached to
    k = NodePairs[i][1];
    counter = 0;
    for (l = 0; l < num_of_node_pairs * NUM_OF_SHORTEST_PATHS; l++)
      if ((ShortestPaths[l][0] == j && ShortestPaths[l][1] == k) ||
            (ShortestPaths[l][1] == j && ShortestPaths[l][2] == k) ||
            (ShortestPaths[l][2] == j && ShortestPaths[l][3] == k) ||
            (ShortestPaths[l][3] == j && ShortestPaths[l][4] == k) ||
            (ShortestPaths[l][0] == k && ShortestPaths[l][1] == j) ||
            (ShortestPaths[l][1] == k && ShortestPaths[l][2] == j) ||
            (ShortestPaths[l][2] == k && ShortestPaths[l][3] == j) ||
            (ShortestPaths[l][3] == k && ShortestPaths[l][4] == j))
          ParticipatingStreams[i][counter++] = l * num_of_wavelengths + num_of_links;
    for (m = 0; m < counter - 1; m++)
      for (count = 0; count < num_of_wavelengths; count++)
          out << "x" << ParticipatingStreams[i][m] + count << " + ";
    for (count = 0; count < num_of_wavelengths - 1; count++)
      out << "x" << ParticipatingStreams[i][m] + count << " + ";
    out << "x" << ParticipatingStreams[i][counter-1] + num_of_wavelengths - 1
      << " = x" << i << ";" << endl;
    ParticipatingStreams[i][num_of_node_pairs * NUM_OF_SHORTEST_PATHS - 1] = counter;
  }

  out << endl;

  /* We require capacities representing wavelengths not to exceed the maximum that can be
       carried out */
  for (i = 0; i < num_of_node_pairs * VarsPerNodePair; i++)
    out << "x" << i + num_of_links << " <= " << WAVELENGTH_CAPACITY << ";" << endl;

  out << endl;


  /* We require capacities representing wavelengths to be non-negative */
  for (i = 0; i < num_of_node_pairs * VarsPerNodePair; i++)
    out << "x" << i + num_of_links << " >= " << 0 << ";" << endl;

  out << endl;



  /* We introduce a constraint in order to avoid overlaps for the same wavelengths of different
      streams that traverse the same link */

  for (i = 0; i < num_of_links; i++) {
    counter = ParticipatingStreams[i][num_of_node_pairs * NUM_OF_SHORTEST_PATHS - 1];
    for (j = 0; j < num_of_wavelengths; j++) {
      for (m = 0; m < counter - 1; m++)
              out << "x" << ParticipatingStreams[i][m] + j << " + ";
      out << "x" << ParticipatingStreams[i][counter-1] + j << " <= "
              << WAVELENGTH_CAPACITY << ";" << endl;
    }
  }

  out << endl;

} // Main



/* This function is used for reading the input file containing information about
   the names of nodes, their location (coordinates) and the traffic corresponding
   to each potential node pair */
void Read_File (ifstream& in, char Nodes[][11],
                            float coordinates[][2], float **traffic_matrix,
                            int **AdjacencyMatrix, int &num_of_links)
{
  int i,j,x,y;
  char ch;

  for (i = 0; i < num_of_nodes; i++) { /* Create the array of Nodes' names
                                                         and coordinates */
    in >> Nodes[i];          // Name of node
    in >> coordinates[i][0];   // x cordinate
    in >> coordinates[i][1];   // y coordinate
  }

  cout << endl << "Coordinates :" << endl;
  for (i = 0; i<num_of_nodes; i++)
```

```
    cout << Nodes[i] << " : x=" << coordinates[i][0] << ", y=" << coordinates[i][1] << endl;

  for (i = 0; i < num_of_nodes; i++) {   // Reading information about the traffic
    for (j = 0; j < num_of_nodes; j++) {
      in >> x;
      in >> y;
      in >> traffic_matrix[i][j];
    }
  }

  num_of_links = 0;

  while (!in.eof()) {
   in >> x;
   in >> y;
   AdjacencyMatrix[x][y] = AdjacencyMatrix[y][x] = 1;
   num_of_links++;
  }
  num_of_links--;

  in.close();

}  // Read_File



/* This function sorts an array of "count" numbers in increasing order,
     using the "Selection Sort" algorithm */
void Sort (int numbers[], int count)
{
  int i,j,t;

  for (i = 1; i < count; i++) {
    t = numbers[i];
    j = i;
    while (numbers[j-1] > t) {
      numbers[j] = numbers[j-1];
      j--;
    }
    numbers[j] = t;
  }

} // Sort


// Initialization of the array of existing paths between a node pair
void Initialize (int **PathsArray, float LengthsArray[MAX_NUM_OF_PATHS])
{
  int i,j;
  for (i = 0; i < MAX_NUM_OF_PATHS; i++) {
    for (j = 0; j < NUM_OF_HOPS + 1; j++)
      PathsArray[i][j] = -1;
    LengthsArray[i] = 0.0;
  }
}  // Initialize


/* This function sorts the array of the existing direct, 2, 3 and 4-hop paths between
   each node pair in increasing order of length, using the "Selection Sort" algorithm */
```

```
void Sort_Paths (int **PathsArray, float LengthsArray[MAX_NUM_OF_PATHS], int count)
{
  int i,j,k;
  float t,f;
  int row[NUM_OF_HOPS + 1];

  for (i = 1; i < count; i++) {
    t = LengthsArray[i];
    for (k = 0; k < NUM_OF_HOPS + 1; k++)
      row[k] = PathsArray[i][k];
    j = i;
    while (LengthsArray[j-1] > t) {
      LengthsArray[j] = LengthsArray[j-1];
      for (k = 0; k < NUM_OF_HOPS + 1; k++)
              PathsArray[j][k] = PathsArray[j-1][k];
      j--;
    }
    LengthsArray[j] = t;
    for (k = 0; k < NUM_OF_HOPS + 1; k++)
      PathsArray[j][k] = row[k];
  }

} // Sort_Paths
```

E.3. Code Creating xfig Pictures for the 9 Central Nodes of EON

```
/*****************************************************************************

Program: drawnet.cc (1.4) Author: M.C.Sinclair, E.S.E. Date: 94/04/17
(modified by D. Gavalas in order to create an xfig picture for the 9 central nodes of EON)

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

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <assert.h>

#define NUMNODES      9
#define GENOTYPELEN   ((NUMNODES * (NUMNODES-1))/2)
#define PICTWIDTH    640
#define PICTHEIGHT   640
#define PICTMARGIN    40
#define CIRCLESIZE     5
#define LINETHICK      2

typedef struct node_s Node;

struct node_s {
  char     *name;
  int      xTextOs;                /* x text offset (pixels) */
  int      yTextOs;                /* y text offset (pixels) */
  double   x;                      /* x and y in cm; 200km = 1 cm  */
  double   y;
};
```

```
Node nodes[] = {
  {"Paris",       10, 10, 13.02, 10.65},
  {"London",     -40,  4,  13.12, 8.71},
  {"Berlin",     -37,  5,  18.31, 8.96},
  {"Milan",      -40,  5,  15.95, 12.91},
  {"Brussels",    10,  5, 14.40,  9.88},
  {"Amsterdam",   10,  5, 14.82, 9.2}, /* y changed from 8.78 */
  {"Prague",     -40,  5, 18.0, 10.44},   /* x changed from 17.36 */
  {"Zurich",      15,  5, 15.98, 11.96},
  {"Luxembourg",  10,  5, 15.06, 10.30},
 };

typedef struct scale_s Scale;

struct scale_s {
  double xOffset;
  double yOffset;
  double xScale;
  double yScale;
};

void drawNet(char* genotype, int gtlen);
void drawNode(int nodeId, Scale* sf);
void drawLink(double x1, double y1, double x2, double y2, Scale* sf);

int main(int argc, char** argv) {
  if (argc != 2 || strlen(argv[1]) != GENOTYPELEN) {
    fprintf(stderr, "usage: %s %d_bit_genotype\n", argv[0], GENOTYPELEN);
    exit(1);
  }

  drawNet(argv[1], GENOTYPELEN);
}

void drawNet(char* genotype, int gtlen) {
  int i, j, k;
  double minX, maxX, minY, maxY;
  Scale sf;

  assert(gtlen == GENOTYPELEN);
  assert(sizeof(nodes)/sizeof(Node) == NUMNODES);

  /* Calculate scaling factors (reversing y-axis) */

  assert(NUMNODES >= 2);

  minX = maxX = nodes[0].x;
  minY = maxY = nodes[0].y;
  for (i = 1; i < NUMNODES; i++) {
    if (nodes[i].x < minX)
      minX = nodes[i].x;
    else if (nodes[i].x > maxX)
      maxX = nodes[i].x;
    if (nodes[i].y < minY)
      minY = nodes[i].y;
    else if (nodes[i].y > maxY)
      maxY = nodes[i].y;
  }
  sf.xScale  = (PICTWIDTH -(2*CIRCLESIZE)-(2*PICTMARGIN))/(maxX-minX);
  sf.yScale  = (PICTHEIGHT-(2*CIRCLESIZE)-(2*PICTMARGIN))/(maxY-minY);
  sf.xOffset = -minX+((CIRCLESIZE+PICTMARGIN)/sf.xScale);
  sf.yOffset = -minY+((CIRCLESIZE+PICTMARGIN)/sf.yScale);

#ifdef DEBUG
  fprintf(stderr, "sf = (%f, %f, %f, %f)\n",
       sf.xOffset, sf.yOffset, sf.xScale, sf.yScale);
#endif

  /* Initial picture instructions */

  printf("#FIG 2.1\n80 2\n");

  /* Draw Nodes */

  for (i = 0; i < NUMNODES; i++)
    drawNode(i, &sf);

  /* Draw Links */

  for (i = 0, k = 0; i < NUMNODES; i++)
    for (j = i+1; j < NUMNODES; j++, k++) {
     if (genotype[k] != '0')
     /* Link exists between (i,j) */
     drawLink(nodes[i].x, nodes[i].y, nodes[j].x, nodes[j].y, &sf);
    }

  assert(k == GENOTYPELEN);
}

void drawNode(int nodeId, Scale* sf) {
  double x = nodes[nodeId].x;
  double y = nodes[nodeId].y;
  char name[4];
  int i;
  x += sf->xOffset;
  x *= sf->xScale;
  y += sf->yOffset;
  y *= sf->yScale;
  printf("1 3 0 1 0 0 0 21 0.0 1 0.0 %.0f %.0f %d %d %.0f %.0f %.0f %.0f\n",
        x, y, CIRCLESIZE, CIRCLESIZE, x, y, x, y+CIRCLESIZE);
  for (i=0; i<3; i++)
    name[i] = toupper(nodes[nodeId].name[i]);
  name[i] = '\0';
  printf("4 0 28 12 0 0 0 0.0 4 15 33 %.0f %.0f %s´ \n",
      x+nodes[nodeId].xTextOs, y+nodes[nodeId].yTextOs, name);
}

void drawLink(double x1, double y1, double x2, double y2, Scale* sf) {
  /*
    #FIG 2.1
    80 2
    1 3 0 1 -1 0 0 21 0.00000 1 0.000 214 89 10 10 214 89 214 99
    1 3 0 1 -1 0 0 21 0.00000 1 0.000 374 164 10 10 374 164 374 174
    2 1 0 2 -1 0 0 21 0.000 -1 0 0
    214 89 374 164 9999 9999
    */
```

```
x1 = (x1 + sf->xOffset) * sf->xScale;
y1 = (y1 + sf->yOffset) * sf->yScale;
x2 = (x2 + sf->xOffset) * sf->xScale;
y2 = (y2 + sf->yOffset) * sf->yScale;
printf("2 1 0 %d 0 0 0 0 0.0 -1 0 0\n  %.0f %.0f %.0f %.0f 9999 9999\n",
    LINETHICK, x1, y1, x2, y2);
```