Optimization Techniques for the Dimensioning and Reconfiguration of MPLS Networks

Sergio Ariel Beker

Thèse

présentée pour obtenir le grade de docteur

de l'Ecole Nationale Supérieure des Télécommunications

Spécialité : Informatique et Réseaux

Sergio BEKER

Techniques d'Optimisation pour le Dimensionnement et la Reconfiguration des Réseaux MPLS

soutenue le 5 avril 2004 devant le jury composé de

Roberto Sabella (CORITEL)	**Président**
Walid BenAmeur (INT)	
Deep Medhi (UMKC)	**Rapporteurs**
Eitan Altman (INRIA)	
Renaud Moignard (FTR&D)	**Examinateurs**
Daniel Kofman (Télécom Paris)	**Directeur de thèse**

Contact: Sergio Beker <beker@infres.enst.fr>

A Pascale...

Résumé

La superposition de topologies virtuelles à la topologie physique d'un réseau est un des principaux mécanismes de l'ingénierie de trafic. Soit un réseau physique d'une certaine topologie et capacité fixées et une matrice de trafic à véhiculer, il s'agit de trouver une topologie logique permettant de mapper de manière optimale la matrice de trafic sur le réseau physique. A titre d'exemple, la topologie logique peut être bâtie grâce à des connexions ATM ou MPLS; c'est d'ailleurs une des principales raisons de l'intérêt qui est porté de nos jours sur MPLS.

Cette gestion centralisée des ressources peut être vue comme une généralisation du partage de charge, prenant en compte une vision globale du réseau au lie de se limiter à l'ensemble des chemins entre deux points. Mais les deux outils peuvent être vus comme complémentaires si on les regarde à différentes échelles de temps. En effet, changer un layout s'avère plus compliqué que changer une politique de partage de charge. Lors de l'évolution de la matrice de trafic, on peut imaginer qu'on fasse évoluer les politiques de partage de charge afin de pouvoir maintenir le niveau de QoS attendu sans avoir à modifier le layout. Il est clair que sur des échelles de temps plus longues et donc face à des variations importantes de la matrice de trafic, il faudra agir sur le layout.

Nos travaux dans ce domaine ont été réalisés dans le contexte du projet RNRT VTHD et VTHD++. Nous avons apporté deux contributions principales. La première concerne la définition de fonctions de coût que nous considérons mieux adaptées à la réalité d'un opérateur que celles utilisées habituellement, la deuxième concerne la prise en compte du coût de changement d'un layout. Les fonctions de coût usuellement utilisées visent à minimiser certaines métriques telles que le délai. Nous ne considérons que cela soit une bonne approche. A partir d'un certain seuil les applications deviennent insensibles à une réduction supplémentaire du délai de traversée du réseau. Le délai maximum est une contrainte que l'on doit repecter. Il est par contre intéressant d'un point de vu opérateur de réduire les coûts d'opération et maintenance et donc, en particulier, de réduire la complexité des layouts. Celle-ci peut être mesurée comme une fonction du nombre de chemins virtuels qui la compossent (nous parlerons de LSP par la suite même si les idées présentées dépassent largement le cadre MPLS).

Nous avons donc formulé divers problèmes de minimisation de la commplexité des layouts

sous des contraintes de QoS. Il s'agit ici d'une modélisation réaliste mais qui engendre des modèles difficiles à résoudre. En effet, le problème de *Multicommodity Flow* qui en résulte est du type MINLP (Mixed Integer Non-Linear Program). Nous avons d'une part analysé et validé les modèles pour des petits réseaux par des méthodes de résolution deterministes (solver MINLP), et d'autre part développés des heuristiques qui permettent de trouver des solutions proches des optimales pour des réseaux de taille bien plus importante. Nous avons montré que la complexité des layouts peut être significativement réduite en comparaison avec celle obtenue suite à l'optimisation des fonctions de coût classiques.

En ce qui concerne la deuxième famille de contributions, le changement d'un layout implique d'une part un coût d'opération et d'autre part peut engendrer des coupures de service qui affecteront directement le coût d'opération. Nous avons donc formulé une famille de problèmes qui prennent en compte le coût du changement de layout. L'une des heuristiques citées a été adaptée pour analyser ces nouveux problèmes.

Mots Clés : Réseaux Convergents de Nouvelle Génération, Réseaux MPLS, Ingénierie de Trafic, Contrôle Dynamique à Boucle Fermée, Dimensionnement des Layouts MPLS, Reconfiguration des Layouts MPLS, Problèmes MINLP, Heuristiques Approchés, Tabu Search, Algorithme de Déviation de Flots.

Acknowledgments

I would like to express my deep recognition to my thesis director Daniel Kofman. He has supported me through the difficult process of defining the area of research and later bringing those ideas to actual contributions. He has had an infinite patience to discuss with we on and on, even when my ideas weren't clear sometimes. For his unlimited help and dedication through the whole process, for helping me beyond his duties at a human level, my sincere thanks.
Walid BenAmeur and Deepankar Medhi have honored me in taking the responsibility as well as the heavy task of reporting on my thesis, even considering the short delays I have imposed on them. I would also like to thank Eitan Altman, Roberto Sabella and Renaud Moignard for being part of my jury. I highly appreciate the time they have found through their busy agendas to discuss some issues with me, in particular the discussion about solvers with Walid BenAmeur, and whose papers inspired much of this work. Also, some comments from Deep Medhi about methods to solve the formulated problems have been very useful to develop the heuristics presented through this thesis.

A Special thanks to Christian Guillemot from France Telecom R&D, director of the VTHD project in which I've participated, for his patience at the very beginning of my stay in France, when my communication skills in French were nonexistent. To Annie Gravey for her availability and her interest in my activities. To my colleagues in the the VTHD project, for understanding my sometimes limited availability for the project when I was writing the thesis report. In all cases, the discussion about technical matters within and outside the project have enriched me with different points of view about what a Next Generation IP Network is expected to offer.

I would like to express my appreciation and gratitude to the members of the RHD group in the Computers and Networks Department at Telecom Paris: Ramon Casellas and Jean-Louis Rougier for the invaluable information gathered through discussions and innumerable questions. They have always been there for me. Without Ramon Casellas, the implementations of the algorithms proposed through this thesis would have been impossible. I would like also to thank Nicolas Puech and Vasilis Friderikos, members of the AM group in the Computers and Networks Department at Telecom Paris, for their help in defining and implementing the Tabu Search heuristics.

I must also thank my fellow doctoral students at Telecom Paris: Anthony Busson, Ouahiba Fouial and Thomas Quinot, with whom I shared the office and the stress of doing a thesis at different stages of their work and mine.

I gratefully acknowledge Pascale's understanding and support through difficult times. To my parents for their support through time and distance: I'm proud of bringing them this accomplishment.

Contents

Résumé 3

Aknowledgments 6

1. General Introduction 21
- 1.1. Motivations . . . 21
- 1.2. Document Organization . . . 23

2. Technological Context: Next Generation IP Networks 25
- 2.1. Introduction . . . 25
- 2.2. Drivers for Service Integration . . . 25
- 2.3. Next Generation IP Network (NGN) Architectures . . . 26
  - 2.3.1. Definitions and Objectives . . . 26
  - 2.3.2. Transport Layer: Towards an IP Multiservice High Speed Transport 26
  - 2.3.3. Control Layer . . . 28
  - 2.3.4. Service Layer . . . 29
- 2.4. The Role of MPLS in the NGN Transport Infrastructure . . . 30
- 2.5. Conclusions . . . 31

3. Evolved Traffic Engineering 33
- 3.1. Traffic Engineering Objectives and Timescales . . . 33
  - 3.1.1. TE Objectives . . . 34
  - 3.1.2. Control Loops and Timescales . . . 35
  - 3.1.3. The Role of MPLS in The Traffic Engineering . . . 39
- 3.2. Traffic Engineering and Measurements . . . 43
  - 3.2.1. Network States . . . 44
- 3.3. Contributions to the Long-term Control Loop: Dimensioning and Reconfiguration . . . 46
  - 3.3.1. Defining an Optimal Point of Operation: Network Dimensioning . . 47
  - 3.3.2. Varying Traffic Conditions: Network Reconfiguration . . . 48

4. Contribution to the Dimensioning of MPLS Networks: Design of Reduced Complexity Layouts 51
- 4.1. Motivations and Previous Work . . . 51

4.2. Network Model Notation . . . 52
4.3. Building the Cost Functions . . . 56
4.4. Setting QoS Guarantees . . . 58
4.5. Formulation: Minimum Path Set and Flow Allocation Problem (MPSFAP) 60
4.6. Preliminary Results on MPSFAP and Model Validation . . . 62
4.6.1. Traffic Matrixes . . . 63
4.6.2. Network Topologies . . . 63
4.6.3. Reference Problem . . . 64
4.6.4. Interface to Solvers: Modeling Language . . . 64
4.6.5. Results and Analysis . . . 65
4.7. Extensions for Multiple Classes of Service . . . 70
4.8. Conclusions . . . 72

5. Contribution to the Development of Heuristics for Solving the Minimum Path Set and Flow Allocation Problems (MPSFAP) 75
5.1. Exact Methods for Solving MINLP Problems . . . 75
5.2. Meta-Heuristics: Tabu Search Methods . . . 77
5.3. A Tabu Search Heuristic Approach Applied to the MPSFAP Problems . . . 79
5.3.1. Initial Solution . . . 79
5.3.2. Perturbation mechanism . . . 79
5.3.3. Evaluation Functions . . . 80
5.3.4. Evaluation of TS Heuristics for MPSFAP . . . 85
5.4. Ad-Hoc Heuristics Based on The Flow Deviation Algorithm . . . 89
5.4.1. The Flow Deviation Algorithm . . . 89
5.5. A Modified Flow Deviation (MFD) Algorithm for Solving the MPSFAP Problems . . . 91
5.5.1. Path Delay Constraints Simplification . . . 93
5.5.2. Path Weights Calculation . . . 94
5.5.3. Determining the Shift Direction . . . 96
5.5.4. Determining the Shift Factor . . . 98
5.5.5. Determining the Best Node Pair . . . 99
5.5.6. Evaluation of the MFD Algorithm for MPSFAP . . . 101
5.6. Conclusions . . . 104

6. Results and Analysis of the MPSFAP Problems for Large Networks Using Tabu Search (TS) and Modified Flow Deviation (MFD) Methods 107
6.1. Considered Network Topologies . . . 107
6.2. Comparison from TS and MFD for the MPSFAP on Large Networks . . . 108
6.3. Result Analysis of Large Networks by Using MFD . . . 114
6.4. Conclusions . . . 119

7. Contribution to Reconfiguring MPLS Networks: Design of Reduced Reconfiguration and Complexity Layouts 121
7.1. Motivations and Previous Work . . . 121
7.2. Network Model Notation: Extensions for Dynamic Traffic Conditions . . . 123
7.3. Building the Cost Functions . . . 125
7.4. Setting QoS Guarantees . . . 127
7.5. Formulation: Minimum Reconfiguration, Path Set and Flow Allocation Problem (MRPSFAP) . . . 127
7.6. Preliminary Results on MRPSFAP and Model Validation . . . 128
7.6.1. Traffic Matrixes: Considering the Traffic Dynamics . . . 129
7.6.2. Results and Analysis . . . 129
7.7. Extensions for Multiple Classes of Service . . . 135
7.8. Conclusions . . . 136

8. Contribution to the Development of Heuristics for Solving the Minimum Reconfiguration and Path Set Flow Allocation Problem (MPRSFAP) 137
8.1. Selecting Efficient Heuristics . . . 137
8.2. Adapting the Modified Flow Deviation Algorithm (MFD) for Solving the MRPSFAP Problem . . . 138
8.3. A Modified Flow Deviation Algorithm for Reconfiguration (MFD-R) . . . 139
8.3.1. Initialization: The Initial Layout . . . 139
8.3.2. Search for a Shift Direction: Progressive Exploration . . . 141
8.4. Evaluation of the MFD-R Algorithm for MRPSFAP . . . 142
8.4.1. Evaluation of the MFD-R Algorithm on Small Networks . . . 142
8.4.2. Results on Reconfiguration from the MFD-R Algorithm on Large Networks . . . 145
8.5. Conclusions . . . 149

General Conclusions 155

A. Multicommodity Flow Problems 159
A.1. Introduction . . . 159
A.2. Node Formulation . . . 159
A.3. Path Formulation . . . 160
A.4. Solution Approaches . . . 161
A.4.1. Optimality Conditions . . . 162

B. AMPL: A Modeling Language for Mathematical Programming 165
B.1. Mathematical Programming . . . 165
B.2. AMPL Basics . . . 166
B.3. Model Files . . . 167

B.3.1. Model File for MPSFAP1 Problem 167
B.3.2. Model File for MPSFAP2 Problem 169
B.3.3. Model File for MTDFAP Problem 171
B.3.4. Model File for MRPSFAP Problem 172
B.4. Data Files 174
B.4.1. Data File for MPSFAP and MTDFAP Problems 174
B.4.2. Data File for MRPSFAP Problem 175

. **Bibliography** **179**

List of Figures

2.1. NGN Architecture 27
2.2. NGN Simplified Control Architecture 28
2.3. NGN Protocol Structure 31

3.1. The Traffic Engineer and Timescales 36
3.2. Policy Based Network Management Architecture 37
3.3. TE Mechanisms and Their Relationship to Timescales 38
3.4. Closed Loop Traffic Engineering System 39
3.5. Diffserv PHB y PHB Scheduling Class 42
3.6. MPLS-Diffserv LSP Types 43
3.7. TE System and Network States 44

4.1. Path Flows: Relationship to Demands and Link Capacities 55
4.2. Example on the importance of avoiding path-delay constraints on unused paths 59
4.3. NET1 and NET2 Networks. 64
4.4. Average End-to-end Path Delay for NET1 67
4.5. Maximum End-to-end Path Delay for NET1 68
4.6. Average End-to-end Path Delay for NET2 69
4.7. Maximum End-to-end Path Delay for NET2 69

5.1. Results from MINLP Solver (NEOS) and Tabu Search (TS) for NET1 . . . 87
5.2. Results from MINLP Solver (NEOS) and Tabu Search (TS) for NET2 . . . 88
5.3. Modified Flow Deviation Algorithm: Path Delay Constraint Simplification . 95
5.4. MFD Algorithm Example: Overloaded Path is Still the Shortest Path . . . 96
5.5. Results from MINLP Solver (NEOS), Tabu Search (TS) and Modified Flow Deviation (MFD) for NET1 103
5.6. Results from MINLP Solver (NEOS), Tabu Search (TS) and Modified Flow Deviation (MFD) for NET2 104

6.1. VTHD Network Topology 108
6.2. NSF Network Topology 109
6.3. Value of The Objective Function for MPSFAP1 (Total Hops) for VTHD and NSF Network Topologies using TS and MFD Algorithms 112

6.4. Used Paths for Layouts Obtained from MPSFAP1 for VTHD and NSF Network Topologies using TS and MFD Algorithms . . . 113
6.5. Maximum Path Delay in Layouts Obtained from MPSFAP1 for VTHD and NSF Network Topologies using TS and MFD Algorithms . . . 114
6.6. Maximum Path Delay and Average Maximum Path Delay for VTHD Network Topologies Using MFD Algorithm, as a Function of Delay Constraint . . . 115
6.7. Maximum Path Delay and Average Maximum Path Delay for NSF Network Topologies Using MFD Algorithm, as a Function of Delay Constraint . . . 115
6.8. Average Quantity of Hops for VTHD Network Topologies Using MFD Algorithm, as a Function of Delay Constraint . . . 116
6.9. Average Quantity of Hops for NSF Network Topologies Using MFD Algorithm, as a Function of Delay Constraint . . . 116
6.10. Average Quantity of Paths for VTHD Network Topologies Using MFD Algorithm, as a Function of Delay Constraint . . . 117
6.11. Average Quantity of Paths for NSF Network Topologies Using MFD Algorithm, as a Function of Delay Constraint . . . 117
6.12. Feasibility Rate for VTHD Network Topologies Using MFD Algorithm, as a Function of Delay Constraint . . . 118
6.13. Feasibility Rate for NSF Network Topologies Using MFD Algorithm, as a Function of Delay Constraint . . . 118

7.1. Traffic measures at packet level for different timescales [68] . . . 124
7.2. Weekly and Daily Traffic Profile on a OC192 Link [69] . . . 124
7.3. Overall process to generate the test set of demand matrixes (static and dynamic) 129
7.4. Overall process to compare layouts obtained from MPSFAP1 and MRPSFAP for a series from an original matrix . . . 130
7.5. Used paths and accumulated path additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1 . . . 131
7.6. Used hops and accumulated hop additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1 . . . 131
7.7. Path additions and deletions for MPSPAF1 and MRPSFAP for demand matrix series M_1 . . . 132
7.8. Hop additions and deletions for MPSPAF1 and MRPSFAP for demand matrix series M_1 . . . 132
7.9. Used paths and accumulated path additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_2 . . . 133
7.10. Used hops and accumulated hop additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_2 . . . 133
7.11. Path additions and deletions for MPSPAF1 and MRPSFAP for demand matrix series M_2 . . . 134

7.12. Hop additions and deletions for MPSPAF1 and MRPSFAP for demand matrix series M_2 134

8.1. Used hops and accumulated hop additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1 using MINLP Solver (NEOS) and MFD-R Heuristic 142
8.2. Used paths and accumulated path additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1 using MINLP Solver (NEOS) and MFD-R Heuristic 143
8.3. Hops added and deleted for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1 using MINLP Solver (NEOS) and MFD-R Heuristic . . . 144
8.4. Paths added and deleted for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1 using MINLP Solver (NEOS) and MFD-R Heuristic . . . 144
8.5. Active hops and accumulated hop additions for MPSFAP1 and MRPSFAP on VTHD for demand matrix series M_1^{VTHD} using MFD and MFD-R heuristics 145
8.6. Active paths and accumulated path additions for MPSFAP1 and MRPSFAP on VTHD for demand matrix series M_1^{VTHD} using MFD and MFD-R heuristics 146
8.7. Hops added and deleted for MPSFAP1 and MRPSFAP on VTHD for demand matrix series M_1^{VTHD} using MINLP Solver (NEOS) and MFD-R heuristics 146
8.8. Paths added and deleted for MPSFAP1 and MRPSFAP on VTHD for demand matrix series M_1^{VTHD} using MINLP Solver (NEOS) and MFD-R heuristics 147
8.9. Active hops and accumulated hop additions for MPSFAP1 and MRPSFAP on NSF for demand matrix series M_1^{NSF} using MFD and MFD-R heuristics 147
8.10. Active paths and accumulated path additions for MPSFAP1 and MRPSFAP on NSF for demand matrix series M_1^{NSF} using MFD and MFD-R heuristics 148
8.11. Hops added and deleted for MPSFAP1 and MRPSFAP on NSF for demand matrix series M_1^{NSF} using MINLP Solver (NEOS) and MFD-R heuristics . 148
8.12. Paths added and deleted for MPSFAP1 and MRPSFAP on NSF for demand matrix series M_1^{NSF} using MINLP Solver (NEOS) and MFD-R heuristics . 149

List of Tables

2.1. Drivers for Service Integration 25

4.1. Example on the importance of avoiding path-delay constraints on unused paths 60
4.2. Quantity of Hops and Paths for MTDFAP, MPSFAP1 and MPSFAP2 Problems with NET1 Network 66
4.3. Quantity of Hops and Paths for MTDFAP, MPSFAP1 and MPSFAP2 Problems with NET2 Network 68

5.1. Quality of the Solutions from TS with respect to MINLP for MPSFAP1 and MPSFAP2 86
5.2. Comparison of Average Quantity of Hops from MINLP and TS for MPSFAP1 and MPSFAP2 86
5.3. Comparison of Average Quantity of Paths from MINLP and TS for MPSFAP1 and MPSFAP2 88
5.4. Quality of the Solutions from MFD with respect to TS and MINLP for MPSFAP1 102
5.5. Quantity of Paths and Hops for the Solutions from MINLP, TS and MFD for MPSFAP1 102

6.1. Average Execution Times and Feasibility Ratios for MPSFAP1 by Using TS and MFD Algorithms 110
6.2. Average Values of the Objective Function for MPSFAP1 by Using TS and MFD Algorithms 110
6.3. Average Paths and Average Maximum End-to-end Path Delay for MPSFAP1 by Using TS and MFD Algorithms 111

List of Algorithms

5.1. Pseudo-code for the General Tabu Search Method 78
5.2. Tabu Search Heuristics Applied to MPSFAP: algorithm to generate a neighbor of the current solution 83
5.3. Tabu Search Heuristics Applied to MPSFAP: algorithm to change path flows for a selected node pair 83
5.4. Pseudo-code for the Tabu Search Heuristics Applied to the MPSFAP Problems 85
5.5. Pseudo-code for the Modified Flow Deviation Method Applied to the MPSFAP1 Problem 94
5.6. Pseudo-code for the Improvement to the MFD Method to Consider All Candidate Paths 97
5.7. Pseudo-code for Calculating the Path Weights 97
5.8. Pseudo-code for Increasing the Path Weights for Overloaded Paths 98
5.9. Pseudo-code for Calculating the Shift Direction 99
5.10. Pseudo-code for Calculating the Shift Factor 100
5.11. Pseudo-code for Calculating the Choice Index 101

8.2. Pseudo Code for Calculating the Set of Shortest Paths Within a Path Set . 139
8.1. Pseudo-code for the Modified Flow Deviation Method Applied to the MRPSFAP Problem 140
8.3. Pseudo Code for Examining All Possible Candidates for a Shift Direction Within a Given Set of Paths 141

1. General Introduction

1.1. Motivations

Next Generation IP networks are in their way to become the new paradigm in IP network evolution for telecom operators. In the last decade we have seen the IP architecture and its related protocols dominate over other transport technologies, becoming the natural choice for service integration. Increasing computing power on terminals at lower costs allows for a wide range of capabilities and service offerings, constituting a powerful driver for service integration. At the same time, ever increasingly competitive markets ask for operator's efficiency at economical and technical levels. This need for efficiency pushes the telecom service providers to operate their networks in a reliable and efficient way, constituting another powerful driver for service integration. Indeed, a unified transport infrastructure allows for CAPEX (Capital Expenditures) and OPEX (Operational Expenditures) savings for the operator.

It is not a secret why IP has become the transport technology of choice after the commercial explosion of the Internet. Its simplicity and openness have boosted a wide range of services to be developed everywhere, constituting a positive network externality difficult to overcome by any other transport technology. Despite the reasons that contributed to impose IP as the ubiquitous technology in today networks, it has not been conceived to offer all the transport functionalities that a reliable and efficient service integration requires. IP best-effort (sometimes called least-effort) packet delivery policy and its connectionless nature don't contribute to devise a good Quality of Service (QoS) provisioning scheme, while making difficult for the operator to realize an efficient resource utilization. If IP is called to be the transport technology of use for next generation multiservice networks, we need to add some mechanisms in order to allow for Traffic Engineering (TE) objectives to be realized.

Traffic Engineering for IP networks assembles a set of mechanisms applied at different timescales and points in the network in order to ensure some QoS guarantees to the customer, while efficiently using the available network resources. Provided that the agreed QoS guarantees are met, an efficient use of network resources allows the operator to reduce operations and maintenance (OAM) costs as well as capital expenditures, through careful planning and dimensioning.

The traffic engineering as a way of controlling the dynamic behavior of the network in order to adapt to changing operational conditions (i.e. topology changes, load conditions, etc.) can be thought as a closed loop control system. The different TE mechanisms are appropriate only when applied within a particular timescale and scope. Each timescale corresponds to a control loop. We identify three main timescales, summarized in three main control loops: the long term, the medium term and the short term. In the long term control loop, TE mechanisms are assimilated to the tasks of planning and dimensioning the network: the objective is to position the network in an operational state which is optimal with respect to economical and performance objectives according to the operator's point of view. From the optimal operational state, the control loop will act to adapt the network to the changing operational conditions in its own timescale. The network will have then to be *redimensioned*, leading to the need of *reconfiguring* the network. The *observation* of the operational conditions in order to determine the current network *state* is a main function in the control loop. The inference of the network state from observation need to be related to the timescale in which the control loop is operating. The same principle is applied to the inner control loops, where TE mechanisms are applied in shorter timescales to control the dynamic behavior of the network on less global scopes.

The present work focuses on the dimensioning and reconfiguration aspects of network operation in a medium to long term timescale. Considering that during the planning phase the operator has already optimized the node and capacity placement, the dimensioning phase will optimize the way the paths are set up, as well as the corresponding flow allocation over those paths in order to meet the presented demands with respect to an objective function. The objective functions generally used through the literature aim at optimizing some measure of performance as cross-network delay or link utilization. We find that the classic objectives are not representative of the cost models associated with realistic network operation and maintenance. The first contribution of the present work is the definition of objective functions modeling the actual costs associated with the dimensioning phase. When traffic dynamics is considered, the layout (i.e. the set of paths and the corresponding flow allocation constituting the optimal routing) will have to change accordingly in order to keep the optimality with respect to the design objectives at every significant change in the traffic demands. The transition from the current operational layout to the next one has also a cost for the operator in terms of spare resources that need to be made available, as well as service disruption times to allow for the changes to be made. In large networks this cost could be considerable. Our second contribution addresses the issue of layout optimization considering the complexity of the new layout, as well as the complexity of the transition. To our knowledge, the resulting *reconfiguration* problem has not been considered so far in the literature regarding the MPLS layout design and optimization.

The minimum cost multicommodity flow problems resulting from the realistic modelling are known to be $\mathcal{NP}$-*complete*, which limits the size of the networks that can be treated through numerical exact methods. Available solvers were used to obtain results for the formulated problems for small networks. These results are useful to validate the proposed cost models, as well as the correctness of the problem formulations. To overcome the tractable network size limitation, heuristic methods were developed to approximately solve the described problems. Heuristics may use general search techniques without any knowledge on the nature of the problem: generally called *meta-heuristics*, or they may use search techniques adapted to the nature of the particular problem being treated. This group of heuristics are called *Ad-Hoc heuristics*, and they make use of the knowledge available on the mentioned problem. Our third contribution constitutes the development and implementation of an algorithm using Tabu Search (TS) techniques (meta-heuristics), and of an algorithm based on the well-known flow-deviation algorithm (ad-hoc heuristics) to solve the dimensioning problem on large networks. Also due to size limitations of the deterministic solvers, we need to develop heuristics to solve the reconfiguration problem on large networks. Our fourth major contribution constitutes the development of such algorithm based on the experience obtained when developing and testing the algorithms to solve the dimensioning problem. This solution is based on a further modification to the flow-deviation method presented for the dimensioning problem, which enables it to handle the reconfiguration problem as well.

1.2. Document Organization

The document is organized as follows:

In Chapter 2 the technological context is given. The Next Generation Convergent IP Networks require a unified transport. Being the IP protocol the natural choice for implementing the transport services such architectures require, Traffic Engineering mechanisms have to be introduced in order to be able to control the behavior of the network and to be able to offer QoS guarantees. In Chapter 3, we introduce the concept of Traffic Engineering as a closed loop control system capable of dealing with network dynamics. The different timescales and scopes in which the TE mechanisms are to be applied are identified. Also, the observation and measurement aspects of the TE system are presented, identifying the network states. Actions to be taken according to each network state and timescale (i.e. TE mechanisms to be used on which network elements) are related to each timescale, identifying possible research activities on each one of them. We identify the long term timescale related problems a important to the set up of an operational point for the network, which will affect the efficiency and the quality of the services offered. Two important aspects to control in the long-term control loop are the dimensioning of the virtual topology and the

reconfiguration of the network when the currently operational network layout has to change. In Chapter 4, the network model is presented, as well as the identified factors that take part in the OAM costs from an operator's standpoint. Suitable cost functions are proposed to optimize the MPLS layout according to those objectives, and the problem mathematically formulated. Also, a deterministic solver is used to obtain a preliminary insight in the model and the cost functions proposed. In Chapter 5, both *meta-heuristics* and *ad-hoc heuristics* are implemented through Tabu Search and Modified Flow Deviation algorithms, capable of dealing with large topologies. The results obtained using both heuristics are compared to those obtained using the deterministic solver in order to conclude on the performance and quality of the results obtained. In Chapter 6, both algorithms are used to obtain results for two large topologies representing nation-wide networks in Europe and United States. Results show that the Modified Flow Deviation algorithm we propose largely outperforms the Tabu Search algorithm. In Chapter 7 the reconfiguration problem is studied. When traffic dynamics is considered, the operational point will need to be recalculated. Even when the new calculated layout is optimized taking into account the OAM cost objectives studied in the dimensioning problem, the cost of the transition between the currently operational layout and the new calculated one may be also high in terms of OAM (spare resources and service disruption time). Our interest is to take into account the dimensioning objectives as well as the reconfiguration objectives when calculating the new layout, in an attempt to include realistic cost of operations and maintenance in the optimization objectives. The cost function is modelled and the problem mathematically formulated. Also, results are obtained from deterministic solvers in order to gain some insight in the interest and correctness of the proposed cost function. Finally, in Chapter 8 we propose a further modification to the algorithm presented in Chapter 5, based on the flow-deviation algorithm, which allow us to obtain results for large network topologies. Through an analysis of the obtained results, we conclude in the General Conclusions on the interest of the proposed cost functions, and we discuss some perspectives opened by the work presented here.

2. Technological Context: Next Generation IP Networks

2.1. Introduction

Service integration in a unique infrastructure has been a major objective for the telecom operators since the inception of packet switched networks. The convergence of multiple services in a unique infrastructure is mainly driven by the CAPEX and OPEX cost reductions for the operator. This integration occurs at different *planes* in the network architecture. In the transport plane, IP appears as the natural choice for integration, as most services are being developed to use IP as the transport protocol, and the Internet success has made of IP the transport protocol used everywhere. The evolution of today IP networks towards an integrated multiservice infrastructure requires the adaptation of IP and its related protocols to provide a service transport capable of offering service differentiation and a flexible adaptation to new ever demanding services through signaling and control functions. The architecture appearing as the natural evolution towards this integrated services infrastructure is know as Next Generation Internet (IP) Networks (NGN). In the present chapter, we first introduce the principal factors driving to the need of service integration in a unique infrastructure, and later a brief description of the network architecture in its different functional planes, including the related protocols and standards where applicable.

2.2. Drivers for Service Integration

The main drivers for service integration can be enumerated as in Table 2.2:

User Drivers	**Operator Drivers**	**Market Drivers**
One-Stop Shopping	Integrated/Increased Service Offering	Reduced entry barriers for new operators
Integrated Billing	Reduced Time-to-Market	Third-party applications
Unified Interface for customer care	**Reduced CAPEX/OPEX**	Increased terminal capabilities
Convergent Applications		Increased Access Bandwidth

Table 2.1: Drivers for Service Integration

From the operator's standpoint, service integration will help in reducing capital and opera-

tional expenditures by multiplexing all services onto a unique infrastructure. This integration is possible through packet multiplexing of the different services. As such, mechanisms to ensure that packets are treated in the network according to the needs of the service they carry, as well as to ensure that enough resources are available in order to allow for the correct behavior of all services must be added.

2.3. Next Generation IP Network (NGN) Architectures

2.3.1. Definitions and Objectives

NGN is a concept for defining and deploying networks, which, due to their formal separation into different layers and planes and use of open interfaces, offers service providers and operators a platform which can evolve in a step-by-step manner to create, deploy and manage innovative services. (ETSI-NGN Starter Group).

Besides the integration of services in a unique infrastructure, the separation in layers and the support for a step-by-step migration from legacy infrastructures are key factors in the road to NGN. The separation in layers is important because of the introduction of open and standardized interfaces, which allow for the realization of services independently of the underlying technology. This opens the possibility for third party content and service providers to develop applications working seamlessly with the network, broadening the service offering and generating more positive network externalities. The openness of the interfaces also allows for the support of the different access technologies and terminal types.

The separation of transport, control and service layers in the NGN architecture is depicted in Figure 2.1. In what follows, we will present a brief description of the evolution path at every layer. A detailed discussion about the other components of the architecture such as the access techniques and terminal capabilities evolution are out of the scope of the present work.

2.3.2. Transport Layer: Towards an IP Multiservice High Speed Transport

Different transport technologies have appeared as strong candidates for the integration at the transport layer. From the evolution of telephone networks, the Integrated Services Digital Network (ISDN) and its broadband version (B-ISDN) with Asynchronous Transfer Mode (ATM) at the transport plane, all were conceived to provide evolved traffic engineering to multiple services with a broad range of requirements using a unified infrastructure. ISDN approache is to switch circuits as needed in order to make fixed bandwidth ($n \times 64$ Kbps circuits) available to applications. As such, the traffic engineering aspect is not required, as the applications have a dedicated circuit or bundles of dedicated circuits to meet the bandwidth

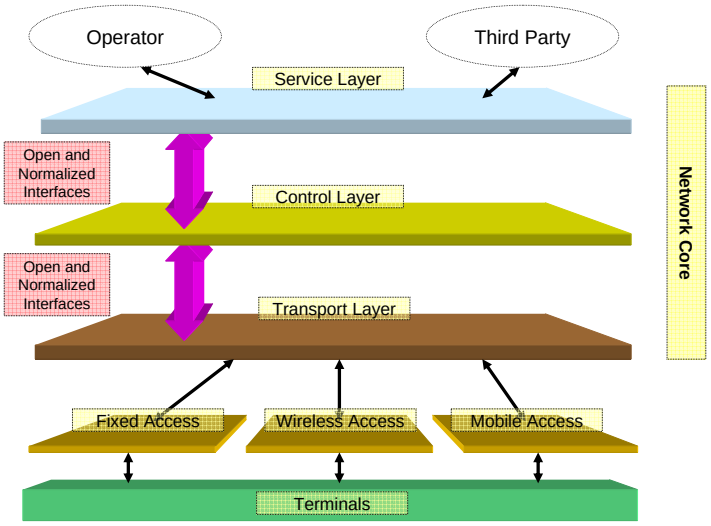


Figure 2.1: NGN Architecture

requirements when needed. On the other hand, B-ISDN with ATM as the transport technology is based on packet switching. *Virtual circuits* offer a bandwidth to the applications, whose characteristics are specified by a *traffic contract* describing the particular treatment given to each one of the predefined *services classes*. ATM was designed to deal with a broad range of service requirements ranging from real-time voice to streaming video. Short equally sized packets allow for a more deterministic packet delay and delay variation when needed. Particular traffic engineering mechanisms (e.g. schedulers, algorithmic droppers, etc.) are required in order to guarantee the appropriate treatment to packets belonging to each service class on a particular virtual connection according to the underwritten traffic contracts. Traffic engineering mechanisms are conceived both to guarantee the QoS specified in the contract and also to control that ingress traffic is complying to the same contract. Despite its potential, ATM didn't succeed in becoming the technology of choice for service integration. In particular, the lack of native ATM services (i.e. designed natively to use ATM as transport technology) is one of the main reasons why it wasn't widely adopted by operators as a unique transport technology capable of integrating voice and data traffic. ATM is today being used mostly in the access to IP broadband networks (e.g. ADSL and cable operators).

Most deployed infrastructures use IP transport technology and related protocols, making unavoidable the use of IP as agent for transport unified infrastructure in NGNs. IP lacks natively of the Traffic Engineering (TE) mechanisms capable of offering a differentiated QoS to the customers, while allowing the operator to efficiently allocate the network resources.

MPLS offers a straightforward way of incorporating such TE mechanisms to IP, enabling evolved network engineering such as optimal dimensioning in both static and dynamic environments.

2.3.3. Control Layer

Among the multiple services that need to be integrated into the Convergent IP infrastructure, real-time services like voice calls and streaming applications (or *media calls* require evolved control functionalities. These *call control* functionalities need to be incorporated into the convergent architecture, and more precisely at the control plane. We briefly describe in this section the main issues to be considered in the NG convergent IP networks architectures, as well as the current efforts in the normalization organizations to solve those issues. The NGN architecture for the control of *media calls* (i.e. voice, video, etc.) is composed of three main elements, which perform the main functionalities needed to both, establish and manage the calls, and interoperate with the legacy networks. The architecture with all the functional elements is depicted in Figure 2.2. The main components in the NGN control architecture are [21]:

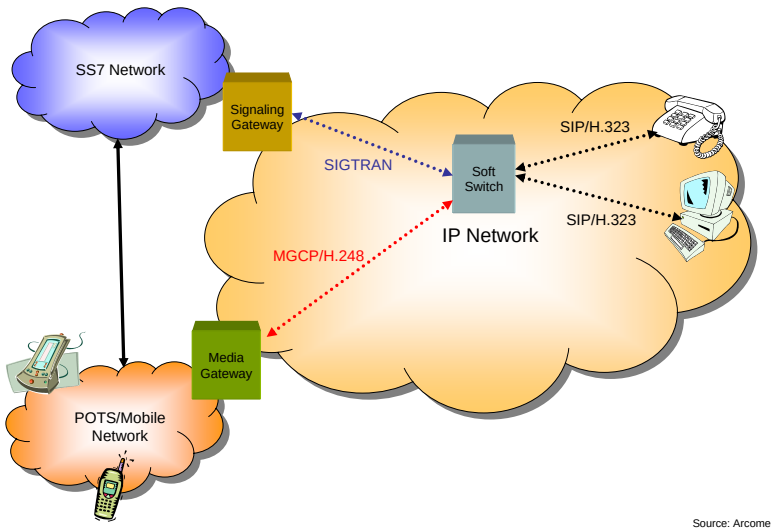


Figure 2.2: NGN Simplified Control Architecture

The Softswitch

Traditional voice switches are replaced in the NGN architecture by the so called *Softswitch* or *Media Gateway Controller*. The softswitch corresponds to the processor and memory

resources in the traditional switch.

The Media Gateway
The interoperation with legacy networks is done at the media gateway. The media gateway is located in the NGN architecture at the media flow transport layer, between the Plain Old Telephone Service network (POTS) or the access network and the IP network. The main functionalities of the media gateway are:

- The coding and packetization of media streams coming from a non IP network, and the conversion of IP packets to the media stream in the destination network.
- The relay of the media streams as signalled by the media gateway controller.

The Signaling Protocols
In order to make possible the convergence of multimedia communications with traditional data services, the NGN network must take care of the media streams at the transport layer. This gives rise to the development of a series of protocols at the control layer. We can classify the different type of protocols in different functional groups:

- **Call Control Protocols:** allowing to establish a media communication from a terminal to another terminal or to a server. Candidates protocols are H.323 (ITU-T) and SIP (IETF).
- **Media Gateway Command Protocols** corresponding to the separation between transport and control layers in the NGN architecture. Allows the media gateway controller to control the media gateways. Candidate protocols are H.248/MEGACO (ITU-T/IETF) and MGCP (IETF).
- **Inter-Media Gateway Controller Signaling Protocols** for the management of the control plane. In the backbone the candidate protocols are Bearer Independent Call Control (BICC) and H.323 both from ITU-T, and SIP-T (IETF). Regarding the interconnection with legacy networks (in particular with SS7 networks), the corresponding signaling gateways implement protocols like SIGTRAN (IETF).

2.3.4. Service Layer

Currently, services are developed within the framework of the corresponding networks: Intelligent Network services (IN) for the telephone terminals (fixed or mobile), and traditional Internet services such as Web, Mail, News, etc. for the IP networks. Together with the evolution of access technologies, which makes more bandwidth available to users, the evolution in terminals capabilities push a transformation of the service platform. This new platform must allow a broad range of users to access services no matter the terminal and protocols

used.

Important aspects of the service offering in the context of the new platform are *adaptability* and *portability* [21]: the user must be able to recover its personal environment no matter what particular terminal he is using, and it should be adapted to that particular terminal. Two basic and complementary models arise:

- **Softswitch Centered:** service architecture based on the OSA/PARLAY normalized service interface. This model is best adapted to *telecom* type services, requiring a strong participation of the call control entities.
- **Web Services Centered:** service architecture based on the protocols and technologies used on the Internet world (XML, SOAP), providing distributed services transported transparently on IP and with a strong participation of the terminals.

2.4. The Role of MPLS in the NGN Transport Infrastructure

As we stated in section 2.3.2, IP has become the natural choice because of its positive externalities, driven mainly by its openness, which has made possible the rapid and effective development of the wide range of services available nowadays. All the characteristics that have made IP the paradigm for service integration are at the same time its drawbacks. IP was conceived to push complexity to the edge of the network: connection oriented transport, checking and recovery of lost packets are functionalities implemented by the hosts participating in the end-to-end communication. The network as such participates with the least effort to convey packets as independent units from one point to the other in the network. Routing and delivery of packets are uncorrelated functions in any network node in an IP network. In order to make IP a QoS aware transport technology, a set of complex traffic engineering mechanisms have to be added. In particular MPLS (Multiprotocol Label Switching) appears as a best adapted complement to IP at level 2 of the OSI layers, representing a good trade-off between complexity of implementation and traffic engineering evolved mechanisms it enables compared to ATM. We will discuss the traffic engineering concepts and mechanisms working together with IP, as well as their timescale and scope of application in Chapter 3.

Most current deployed infrastructures rely on fiber optics as the transmission media. Wave Division Multiplexing (WDM) makes multiple wavelengths available to the upper transmission layers, traditionally organized in digital hierarchies. Time Division Multiplexing (TDM) techniques such as Plesiochronous Digital Hierarchy (PDH) and Synchronous Digital Hierarchy (SDH) provide transmission paths to the IP layer through layer 2 protocols. Other layer 2 transport technologies like Frame Relay (FR) and Asynchronous Transfer Mode (ATM) provide *virtual circuits* to IP, which can be permanently established or on

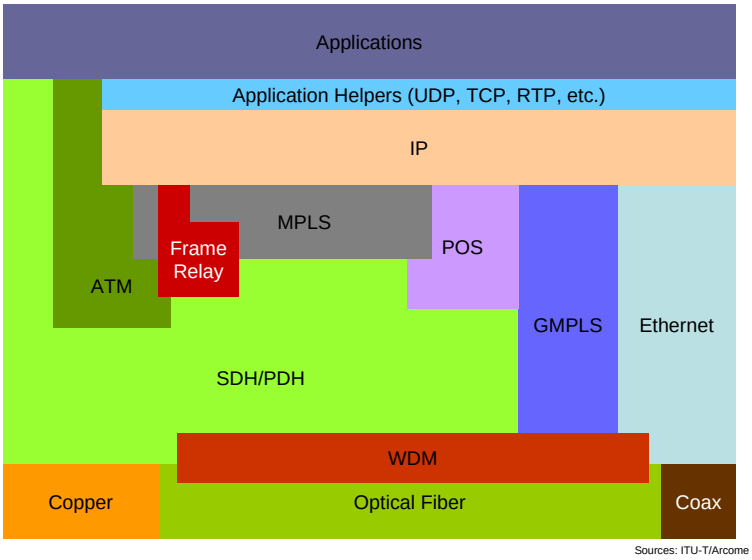


Figure 2.3: NGN Protocol Structure

a demand basis through signaling. More recently, Multiprotocol Label Switching (MPLS) has emerged as the transport technology of choice for network operators. It provides *Label Switched Paths* (LSPs) or *tunnels* individualized by a label added to the IP (or other protocol) packets. The advantage is that packets are switched directly using this label, and no IP address lookup and packet contents processing is further needed while relaying the packet in the network. Only the edge nodes or the end hosts process the IP packet as such. MPLS offers performance gains (due to the switching at layer 2), as well as allows for the implementation of evolved traffic engineering to be added to the IP transport. A generalization of MPLS is currently being defined within many normalization and study organizations and forums (IETF, ITU-T and OIF), which switches wavelengths in the way that MPLS switches packets. We will enumerate and discuss many of its advantages in Chapter 3 in the context of the traffic engineering for the NGNs, as we assume the use of MPLS/GMPLS as the underlying transport technology underneath IP when defining the network models presented in Chapters 4 and 7.

2.5. Conclusions

Service convergence in a unique multiservice infrastructure requires a unified transport layer. IP is the technology of choice to achieve this integration, but it lacks the mechanisms to ensure QoS guarantees to the end users, as well as the mechanisms to allow the network operator to use its resources in a cost-effective way. Complementary traffic engineering

functionalities are then adjoined to IP at the transport layer in order to allow for an end-to-end control of the offered QoS and the resources needed to attain such objectives. Protocols also need to be added at the control layer to deal with signalling of the media calls in the IP world, as well as the interworking with legacy networks. Finally, at the service layer, interfaces must be created to allow users to keep a personalized environment which is both, independent of the terminal and adaptable to its capabilities. In what follows, we will concentrate on the transport layer, particularly in the traffic engineering mechanisms at long-term timescales to allow the operator to attain both objectives: to provide QoS guarantees while reducing operational costs.

3. Evolved Traffic Engineering

In Chapter 2 we established the technological framework: the next generation networks as a multiservice integrated infrastructure. The architectures being proposed deal with service integration at three different levels: transport, control and service layers. Service integration at the transport layer require the adaptation of the existing IP transport services to match the new requirements: providing service differentiation to the different applications, flexibility to easily adapt to new ever developing services, and tools to control the behavior of the network and make an efficient use of network resources. Evolved traffic engineering helps in complementing the current IP transport functionalities in order to achieve those objectives.

Conceptually, the control of the network dynamics can be thought as a feedback control system, including a *demand system* (i.e. the traffic), a *constraints system* (i.e. the interconnected network elements), and a *response system* (i.e. the network protocols and control mechanisms running on the network) [6]. The Traffic Engineering (TE) defines the parameters and points of operation for the network, as well as the mechanisms that control the return of the network to the defined operational points when the *demand system* and/or the *constraint system* vary.

In order to define a closed loop control system we have first to identify the traffic engineering mechanisms available to the operation, as well as the timescale in which they can be used. Then it is necessary to decide when each of those mechanisms can be used either to correctly dimension the network and set a point of operation, or to react to dynamic changes in the traffic conditions. Care must be taken that the interaction among the different TE mechanisms won't produce unwanted effects. In what follows, we present the traffic engineering objectives, identifying the main elements contributing to the TE system.

3.1. Traffic Engineering Objectives and Timescales

To meet the TE objectives, the operator disposes of a bundle of TE mechanisms to control the response of the network to dynamic traffic conditions, and to position it in the desired point of operation. Thinking the dynamic management of the network as a classical control system, the pertinent timescales must be identified as well as the appropriate traffic engineering mechanisms to each timescale. Each timescale is identified with a control loop.

The different TE mechanisms interact, even when applied at different timescales. This interaction could produce negative effects on the control of the network behavior, and must be accounted for when defining the control processes associated to each control loop.

3.1.1. TE Objectives

The network operator aims at providing QoS guarantees to its customers while making an efficient resource usage. As a key element in the control system, these objectives become those of the traffic engineering. The TE objectives as seen by the operator can then be summarized as follows:

Traffic Oriented: related to the control of the QoS provided to the customer traffic. From an end user's standpoint, these objectives can be relative (e.g. service differentiation with relative priorities and drop probabilities at packet or flow level), or absolute (e.g. guaranteed throughput, end-to-end delay, etc). Parameters generally associated to the measure of the traffic oriented objectives include packet loss, end-to-end packet delay, delay variation and throughput among others. From an operator's standpoint, besides providing individual guarantees to the flows per user and application, an important objective affecting the whole network performance is the proportion of total customer traffic meeting the underwritten *Service Level Agreements* (SLAs) with respect to the total traffic transported. This is not perceived individually by end users, but affects the QoS guarantees given to them as a whole.

Resource Oriented: related to the efficient usage of network resources. These objectives are seen exclusively from an operator's standpoint, and are determined mainly by the business model adopted. Traffic oriented objectives can be met, even with poor efficiency on the resource oriented objectives. For instance, currently used policies tend to minimize the traffic engineering efforts by overprovisioning the network, resulting in ever increasing needs for bandwidth to provide QoS guarantees to end users.

A traffic engineering system is called *rational* if it is designed to attain the traffic oriented objectives, while optimizing the resource oriented objectives [6].

The task of the traffic engineering system is to drive the network from sub-optimal states towards optimal states (in terms of both traffic engineering objectives) reflecting the business model and service agreements with its customers. In terms of TE objectives, sub-optimal network states are related to congestion situations produced wether by insufficient network resources for a given demand at a particular instant, or by inefficient mapping of the transported traffic over those resources. The TE system can deal with congestion situations produced by insufficient network resources by increasing the available capacity (planning

and dimensioning) if the traffic injected by the end users conforms the traffic contracts, or by policing the traffic at the ingress if some of those contracts are violated. The objective of traffic policing is to make the ingress traffic conform to the underwritten contracts, and is applied on a customer by customer basis on the ingress interface acting at packet or flow level (e.g. traffic shaping, flow control, packet marking, etc.). New techniques aim at controlling the traffic entering the network through pricing policies, encouraging the customer to reduce the sending rate when congestion situations arise. The TE system must identify the appropriate mechanism to use on each network situation, since the different TE mechanisms can't be applied at the same timescale. For instance, a congestion state produced by a systematic increase in traffic demands cannot be addressed using traffic policing, requiring a network redimensioning (or even a capacity reallocation process) applied in a longer term. On the other hand, a congestion state produced by a temporary increase in the traffic demands for a particular group of customers (and maybe observed to be out of profile according to the traffic contracts) can be addressed by means of short-term related TE mechanisms (e.g. traffic shaping) applied to the particular interfaces associated with those customers. In the last case, a network redimensioning would result impractical.

Congestion states produced by inefficient traffic mappings on the available network resources can be addressed by using routing techniques acting at different timescales. The administrative metrics used by the IGPs in a pure IP environment for instance, can be dynamically modified to adapt to varying traffic conditions; load sharing (Equal Cost Multipath or MPLS based) can be established to allow the network to attain higher transport efficiencies (both applied at short to medium term timescales). On a longer timescale, the base *layout* representing the actual traffic mapping to the network resources can be recalculated, resulting in a more efficient resource usage.

In what follows, we present in detail the timescales that can be identified in a traffic engineering system and their correspondence to the control loops involved. The different TE mechanisms associated to each control loop are presented and discussed.

3.1.2. Control Loops and Timescales

In previous sections we have modelled the traffic engineering as a closed loop control system, capable to identify the network state and take actions to drive the network to a desired operational state. As in any control system, the current state is measured and compared to the desired state. This information or *difference* between both states is used to take a decision about the action to take to reduce that difference. To determine the state of the network, measure and inference from those measures becomes a key element in the system. It will be treated in more detail in section 3.2. In the case of the traffic engineering control

system for IP networks, the available actions are the different TE mechanisms acting at packet and flow levels. Naturally, those mechanisms have to be coordinated and carefully engineered so as to interwork and not to interfere one to another. Figure 3.1 shows the role of the traffic engineer as a decision maker in the process of controlling the network dynamics. Events arise and are detected by the observation and measurement system, the state of the network is inferred from the observed events and the decision system must decide what action is appropriate to the observed state, given the context.

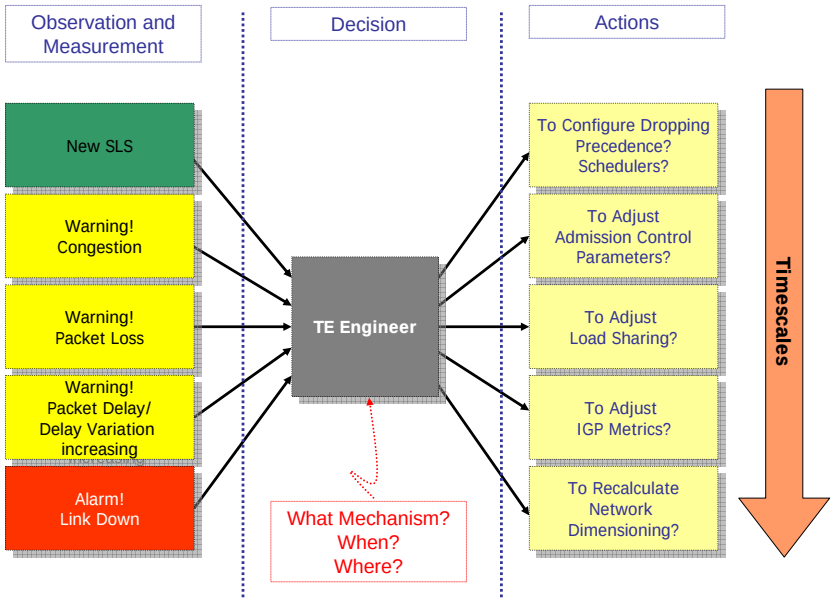


Figure 3.1: The Traffic Engineer and Timescales

The *traffic engineer*, as a decision maker, will take into account the nature of the observed state and decide to apply a particular action or set of actions. The correspondence between a given condition and the action taken is defined as a *policy*. In recent years, the Internet Engineering Task Force (IETF) has made efforts towards defining an architecture capable of dealing with the complexity of the management in an evolved traffic engineering environment: the *Policy Based Network Management* (PBNM). The *Resource Allocation Protocol* (RAP) working group has defined the framework, including the architectural elements [1]. In Figure 3.2, the architectural elements are shown. The *Policy Enforcement Points* (PEPs) (i.e. routers, switches, access points) are the elements where the different TE mechanisms are applied, since they *see* the packets and flows traverse through them and are in a position to change the treatment given to the traffic. The actions to be taken are stored in *policy repositories* [4] in the format of *policies* (i.e. conditions and the corresponding actions). The

Policy Decision Point (PDP) is the element in charge of identifying the adequate policies to apply to what PEPs, and to communicate them through the *Common Open Policy Service* (COPS) protocol [2]. The PBNM architecture is intended to be able to manage any aspect of network operation, from Diffserv and traffic policing to security or billing. In particular, TE mechanisms can be configured through the policy based architecture by means of COPS protocol extensions (e.g. Diffserv [3] or RSVP-TE [5]).

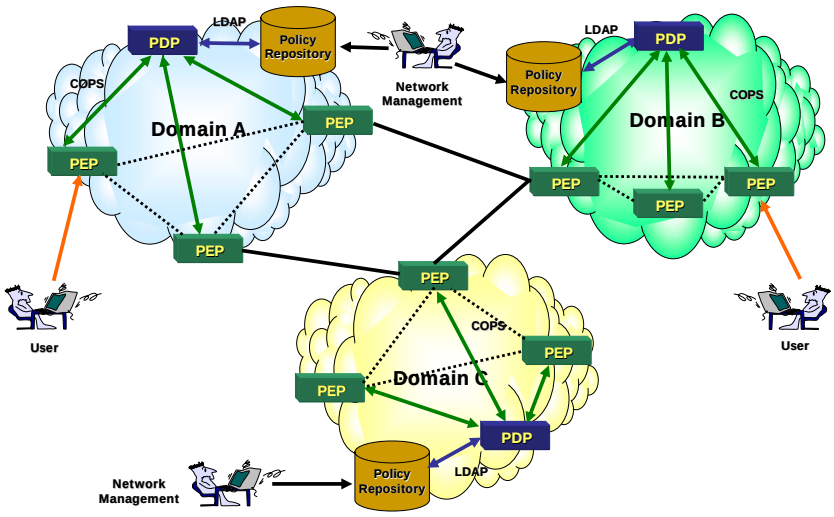


Figure 3.2: Policy Based Network Management Architecture

The PBNM architecture tries to somewhat standardize the interface between the network elements and the measurement system to determine the state of the network, and between the decision maker (i.e. the traffic engineer) and the network elements to implement the mechanisms which will help in driving the network to the desired state. However, the way in which decisions are taken are proprietary to the operator, as they constitute a main source of differentiation among them. As such, the design of *policies* for traffic engineering are left open to research activities. We have already stated that TE mechanisms are appropriate within a given timescale. Figure 3.3 classifies some of the TE mechanisms in relation to the correspondent timescale in which they are generally applied, relating them also to the scope of application. TE mechanisms inscribed in shorter timescales are also likely to be applied at local scopes (e.g. individual interfaces), while TE mechanisms inscribed in longer timescales are likely to be applied at global scopes (e.g. routing areas or the whole network).

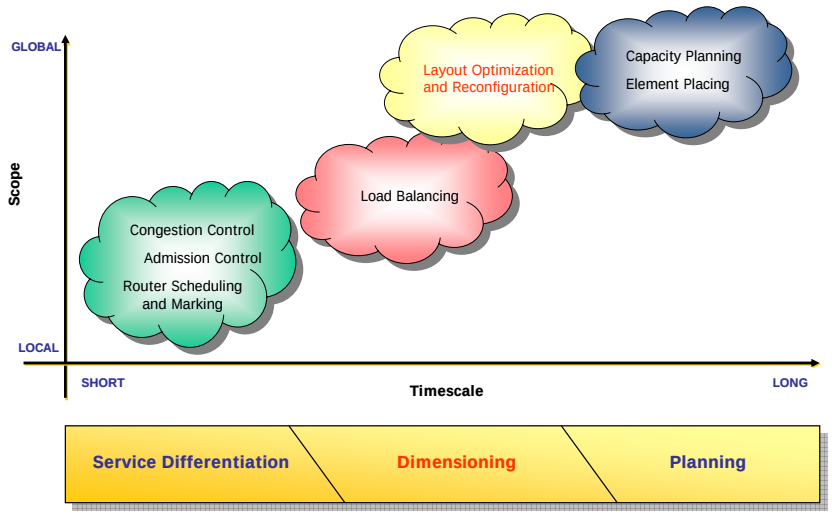


Figure 3.3: TE Mechanisms and Their Relationship to Timescales

To deal with interdependencies among the TE mechanisms, a control loop can be defined corresponding to each of the identified timescales. Nesting the control loops, actions taken in an outer loop generate a network state that is presented to inner loops as the observed state, so interdependencies are solved in the model. The longer timescale can then be identified with the planning and dimensioning control loop: the network will be planned and dimensioned taking into account the traffic forecast and the observed network state, positioning the network in an optimal state with respect to some cost function. This cost function would take into account OAM related and deployment costs to the operator. Figure 3.4 shows the different control loops and some of the actions identified with each one of them.

Once the network positioned in an optimal operational state, significative load variations will require a new network dimensioning (and eventually a new capacity allocation scheme), resulting in a network reconfiguration. Depending on the transport technology being used, the reconfiguration in the network can involve changing the routing policies or setting up a completely new layout (e.g. in the case of MPLS). What dimensioning and reconfiguration policies are to be used give rise to a bunch of research directions in the field of network optimization and operational research. For small load variations, redimensioning and reconfiguring the network may be impractical (i.e. the cost of reconfiguration can be higher than the cost of operating a suboptimal network). Those variations can be dealt with in an

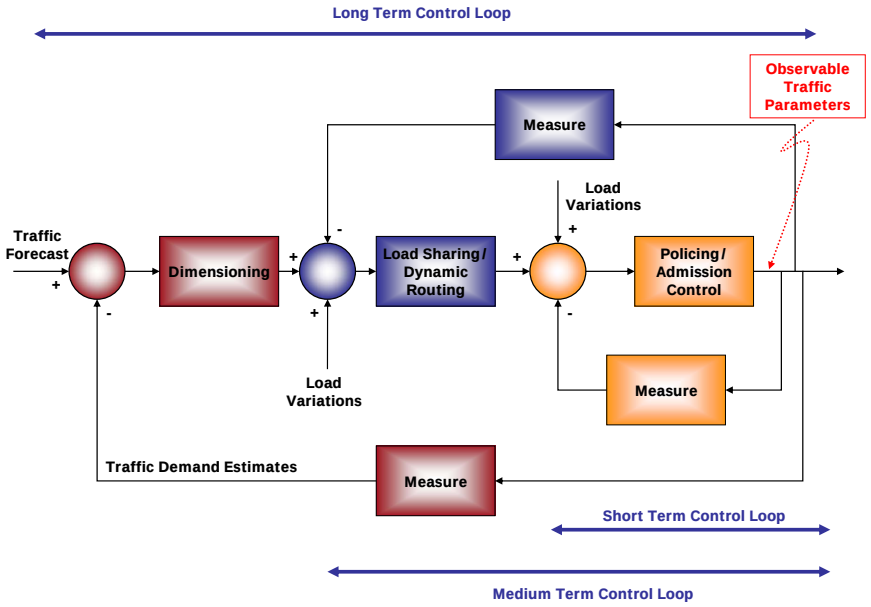


Figure 3.4: Closed Loop Traffic Engineering System

inner control loop corresponding to a medium-term timescale. Techniques such as optimal load sharing [76, 64] in the case of MPLS as transport technology, or changing the metrics of the IGP routing protocols [45] in a pure IP environment, can help to balance the load dynamically and allow for a better efficiency of the network resources. In a shorter timescale, and for traffic variations on a user-by-user flow level, techniques such as flow control at TCP level [77], admission control or traffic shaping among others can be used. Here again, policies on how to decide the techniques to use at medium and short term timescales give rise to a wide field of research. In this thesis, we are interested in the problems associated with the dimensioning and reconfiguration of the network on a long-term timescale, in particular considering MPLS layouts. We consider that many contributions are possible in this field, since small or no attention has been paid to the dimensioning and reconfiguration optimization of layouts for transport techniques other than optical networks, taking into account the OAM costs as viewed by the operator and the particularities of transport technologies associated to the IP family like MPLS.

3.1.3. The Role of MPLS in The Traffic Engineering

The way in which traffic is routed through the network is one of the most important set of tools the operator has to meet the TE objectives. Looking in detail to the network elements in charge of delivering the packets at any point in the network, two steps can be

identified: *packet routing* (also called the *control part* and *packet forwarding*. The IETF has recently created a Working Group (the Forwarding and Control Element Separation *forces* WG to address issues regarding both functionalities). In pure IP environments, IGP routing protocols are generally used within an *autonomous system* to decide the routing of packets through the network. These IGPs (e.g. OSPF, IS-IS) take routing decisions on a per-packet basis, using the destination IP address as the sole information, resulting in a limited capability for the implementation of evolved TE techniques to control the traffic distribution within the network, such as non-ECMP (Equal Cost Multipath) load sharing and general optimal routing virtual topologies among others. Also, since routing decisions are taken hop-by-hop, the packet forwarding part doesn't offer either much room to implement evolved TE functionalities in order to offer end-to-end service differentiation and QoS guarantees. Indeed, the fact that routing decisions are taken on a hop-by-hop basis makes difficult a coordinated packet treatment on all the interfaces used by a particular path for a given service class.

Significant work is currently undergoing to overcome the difficulties in implementing evolved TE capabilities in an IP transport technology. In pure IP environments, the limitations of the IGPs to better control the traffic distribution are being addressed by dynamically changing, for instance, the protocol metrics associated to each link (those used by the protocol to calculate the routes) according to some measure of performance generally associated with congestion [75, 45]. Protocol extensions are being considered also in order to establish routes on demand, based on service classes traffic requirements [78]. Another aspect difficult to implement in a pure IP transport with IGPs is the *load sharing* functionality (i.e. to share the load from source to destination among multiple paths), allowing for a more efficient use of available capacity. IGP load sharing is limited to ECMP: the set of paths on which the load is shared have to be equal cost paths in terms of the protocol metrics and the load is shared equally among them, otherwise routing loops could take place. Regarding the forwarding part of the routing function, the Differentiated Services (Diffserv [79]) and the Integrated Services (Intserv [80]) architectures are models proposed by the IETF to provide relative service class differentiation in the former or absolute QoS guarantees in the later. However, the connectionless nature of IP routing makes difficult to implement end-to-end service differentiation using IGPs, since no global vision of the network is available to the routing protocol at the moment of deciding the route, and packets belonging to the same service class are not guaranteed to receive the same (or equivalent) treatment along the whole path, as the path can change during the life of the flow.

Multiprotocol Label Switching (MPLS) enables the possibility to implement evolved traffic engineering techniques in IP networks. These techniques allow to solve the above mentioned problems associated to the use of IGPs in pure IP environments. MPLS architecture is de-

scribed in [8]. MPLS establishes *virtual tunnels* or label switched paths (LSPs), associating a label to each LSP. A label is associated to a *Forward Equivalent Class* (FEC) (e.g. origin and destination IP address + class of service), which ensures that all packets labelled with a particular FEC will be delivered along the same path. In this way, routes can be decided by the traffic engineering system using more complete network state information, and established at the origin (source routed paths). Also, as the whole path is known and all packets belonging to a FEC are guaranteed to be forwarded over the same path, the treatment given to all packets of that FEC can be guaranteed also to receive the same treatment, easing the implementation of QoS guarantees. The requirements for implementing TE functionalities with MPLS are described in [9]. The routers implementing MPLS are called *Label Switched Routers* (LSRs); the switching of labels instead of looking up to the IP level results in an important efficiency gain in the routers.

One important characteristic in traffic engineering is the concept of *traffic trunk*. A traffic aggregate is defined by the whole traffic going from a particular source to a particular destination, and belonging to a given class of service. The problem of mapping those traffic aggregates on the physical network topology is a main problem in the TE. In an MPLS environment, each traffic aggregate can be directly mapped to a particular LSPs (or multiple LSPs in the case of load sharing). Those paths can be calculated to optimize a given cost function depending on performance and/or economic objectives. The virtual topology or *layout* obtained (corresponding to the *general optimal routing*) can be wholly implemented deploying the adequate labels, which is equivalent to establishing the virtual paths on the network. A particular treatment can be associated to each path, adding QoS guarantees to the traffic traversing the network. In summary, MPLS allows for performance and efficiency optimization on an IP network in many aspects:

Congestion States Produced by a Sub-Optimal Routing : The existent LSPs can be rerouted or new LSPs added in order to redistribute the traffic so as to drive the network towards an optimal state (dimensioning and reconfiguration).

Congestion States Produced by Transitory Traffic Variations : The existent LSPs can be reconfigured to deal with non systematic traffic variations (load sharing).

MPLS support Differentiated Services (MPLS-Diffserv) [11] enables the possibility of engineering the different classes of service defined in the Diffserv architecture by assigning to each *per-domain scheduling class* (PSC) [81] a particular FEC (or set of FECs), which ensures that the appropriate treatment will be given to packets belonging to that PSC. Figure 3.5 shows the service class definition in Diffserv architecture:

PHB Group (PG): a set of one or more *Per-Hop Behavior* (PHB) with a common constraint (queue servicing or management policy).

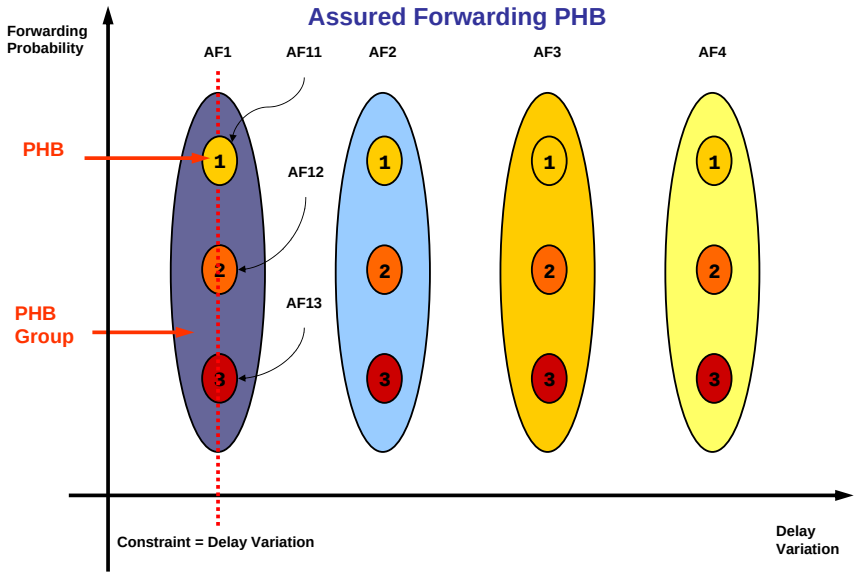


Figure 3.5: Diffserv PHB y PHB Scheduling Class

PHB Scheduling Class (PSC): a PHB Group with the common constraint of ordering preservation for packets belonging to the same microflow.

Figure 3.6 shows the different types of LPS supporting Diffserv PSCs. *EXP-Inferred-PSC LSP* (E-LSP) uses the experimental (EXP) field defined in the MPLS header to signal the PSCs being transported in a particular LSP (defined by the MPLS label). An E-LSP can transport many PSCs, resulting in an efficient use of label space, but a less efficient separation of service classes within a single LSP. A possible use of E-LSPs is, for instance, the support of service differentiation within a single *Virtual Private Network* (VPN): The FEC determines the paths used by the particular VPN to transport the customer traffic, and supporting service differentiation within the VPN. In this case, the TE mechanisms necessary to ensure the service differentiation are applied at the edge of the network, and are based mainly on traffic policing and admission control.

The *Label-Only-Inferred-PSC LSPs* (L-LSP) on the other hand, uses the EXP field and the label in the MPLS header to signal the service class being transported. This means that each LSP transports a particular PSC (e.g. AF1 service class with the 3 dropping precedences AF11, AF12 and AF13), offering a better separation of service classes within the network. Each service class could have, for instance, a dedicated LSP, which offers the possibility of applying TE mechanisms along the path in order to offer individual guarantees. This advantage comes at the price of intensive label space usage.

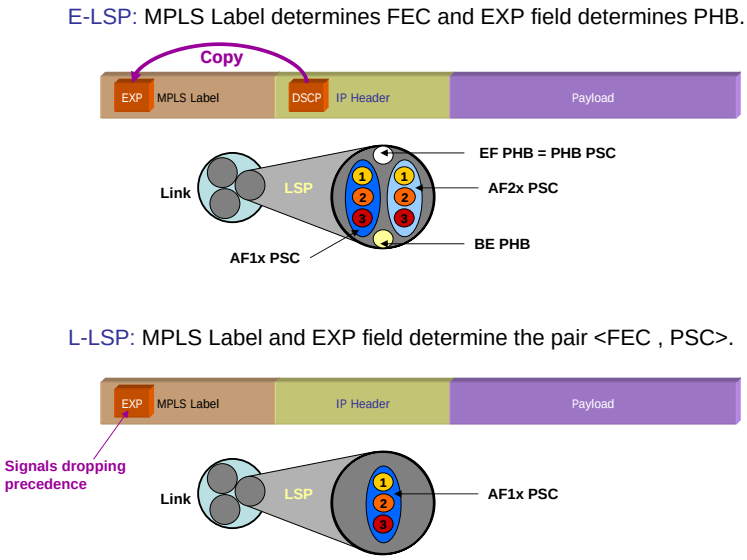


Figure 3.6: MPLS-Diffserv LSP Types

MPLS appears then, as a natural solution to the difficulties found in pure IP environments when implementing evolved TE functionalities. The introduction of MPLS as a transport technology is a key complement to IP to achieve the main objective of service integration in a unique infrastructure, while making an efficient use of network resources. As we mentioned before, MPLS allows the implementation of the general optimal routing layout by establishing source routed paths *copying* the optimally calculated layout. As a consequence, the number of paths (LSPs) required to establish a particular layout could be specially important for large networks, introducing the associated problems of layout complexity and reconfiguration. In [40], the implementation of the general optimal routing layout with a mix of IGP for the compatible part of the layout and MPLS to complete the non-compatible part of the layout is studied. However, the design of reduced complexity layouts and the complexity of layout reconfiguration are open issues scarcely addressed in the literature, and are the main field of contribution in this thesis.

3.2. Traffic Engineering and Measurements

At the beginning of this chapter, we have described the TE system as a closed loop control system, in which the TE mechanisms are the actions taken to drive the network to the

desired operational state, and the observation and measurement subsystem is responsible for determining the current state of the network in order to make possible for the *traffic engineer* to take an appropriate decision. The observation and measurement sub-system is a key element in the traffic engineering conceived as a control of the network dynamics. The observation and measurement sub-system is responsible for determining the state of the network based on measures integrated in the correspondent timescale. Also, based on measures and an appropriate statistical model, it contributes to determine the traffic forecast that can be used in the *capacity planning* and *dimensioning* phases of the network design. In what follows, some preliminary notions on how network states can be classified and observed are introduced.

3.2.1. Network States

Regarding the dynamic control concept of the TE, the network can be thought as a *state machine*, where the states are *optimal* (i.e. desired states) or *sub-optimal*, and the different TE mechanisms are used to transition the network from sub-optimal states to optimal states. Figure 3.7 presents a simplistic representation of network states that can be identified according to observation and measurement within the network, and the related TE mechanisms that can be used to drive the network among states.

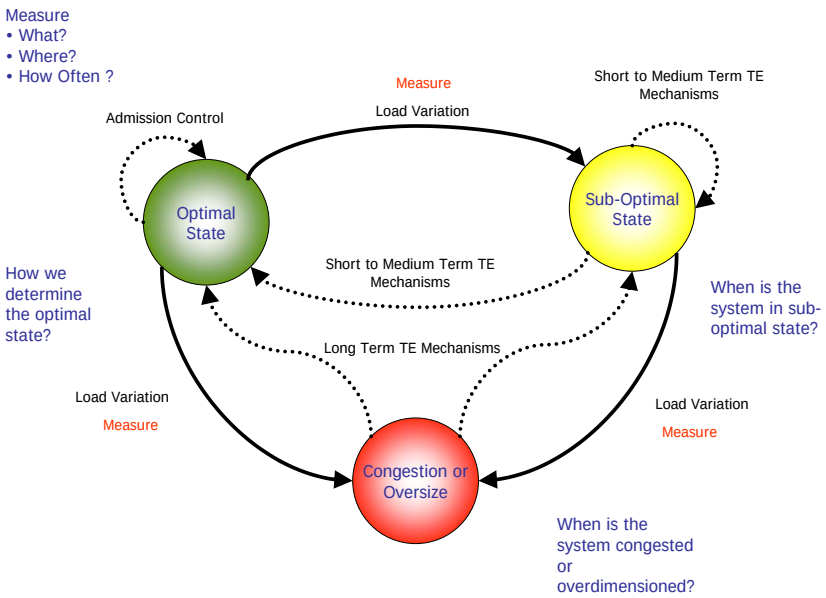


Figure 3.7: TE System and Network States

Optimal State : The network is in a state defined by the operator as the optimal state with respect to the cost functions representing its business model. In the optimal state, all the underwritten service contracts (SLAs) are honored, while efficiently using the network resources according to the operator's objectives.

Sub-Optimal State : The network is in a state defined by the operator as sub-optimal with respect to its business objectives. In a sub-optimal state, the service contracts are still honored, but the network resources are no longer being used efficiently. Sub-optimal states are generated by transitory variations in the load conditions.

Congestion or Sub-utilization State : The network is in a congestion state when service contracts are not honored in a systematic way, even when network resources are efficiently used (although this is not likely). A congestion state is produces by a systematic and significant variation in traffic load conditions (traffic increase). If service contracts are honored, but the network resources are being systematically used in an inefficient way, we identify the network state as a sub-utilization state. TE techniques like dimensioning and reconfiguration allow the operator to return to the optimal state in either case.

Each network state is univocally defined by a set of observable and/or measurable parameters. The observation and measurement sub-system is the one responsible to infer the network state from these observations and measurements. The observation implies two different levels:

- Detailed: per customer and service class. It individually observes (through measurement) the state of the individual service contracts.
- Global: on a traffic aggregate basis within the network. It helps the operator to determine if the network resources are correctly provisioned so as to ensure the QoS to the set of service contracts.

The combination of detailed and global observation is used by the sub-system to determine the network state, and again, this is based on the policies configured by the operator, representing mainly its business model. As such, the particular parameters and conditions that determine the network state are proprietary to each operator, and constitute a source for market differentiation.

Observation and measurement in the network can be *passive* or *active*. Passive measurement is based on the collection of information in the network without injecting measurement traffic, while active measurement is based on special packets injected for measurement purposes. Four main approaches have been considered [82] for gathering information from the network passively:

1. The necessary traffic statistics may be available directly from Single Management Protocol (SNMP) Management Information Bases (MIBs), depending on the forwarding technology being used (e.g. MPLS MIBs can be consulted to obtain the measured traffic volume on the LSPs).

2. The demands can be computed by combining packet level or flow level measurements at the edge of the network with the information available in routing tables.

3. The demands can be inferred based on observation of the aggregate loads inside the network in conjunction with routing data (this approach is known as *network tomography*).

4. New techniques for packet sampling offer the possibility of direct observation of the demands as it flows through the network.

Important work is undergoing at the IETF regarding the active metrology in IP environments. The IP Performance Metrics (IPPM) working group defined the framework architecture [83], in which a set of metrics to measure different aspects of network performance such as connectivity, one way delay and one way packet loss among others are defined. Important research directions are being addressed regarding active measurement, such as determining the sampling frequency, statistic techniques to interpret the measures among others.

3.3. Contributions to the Long-term Control Loop: Dimensioning and Reconfiguration

The long term control loop function is to establish an operational point for the network. This operational point is realized by the set of paths and the corresponding traffic being routed over that set of paths or *layout*. The main objective is to efficiently route the traffic demands between every pair of nodes in the network. To obtain an optimal layout according to the operational and business objectives is one of the main problems the operator has to face. The problem of designing this optimal layout is referenced as *dimensioning*. Once this operational point established, and according to the measured traffic dynamics and the operator's service and business model, this operational point will have to be recalculated. The repositioning of the operational point will require a reconfiguration of the set of paths and and the flow distributions over those new paths. The resulting reconfiguration, depending on the size of the network and the transport technology being used, could imply a large amount of resources being involved in the change, resulting in a high cost of operation (maybe higher than operating the network on a sub-optimal point). A new objective associated with the dimensioning arises: the optimization of the reconfiguration. In our work, we address both, the dimensioning and reconfiguration problems taking int account the OAM

costs as seen by the operator. Increasing attention is being paid to these problems, initially treated in relation to the design of optical transport networks, as the service integration at the IP transport layer requires the introduction of technologies like MPLS.

3.3.1. Defining an Optimal Point of Operation: Network Dimensioning

As it was described in previous sections, in a pure IP environment the traditional routing protocols calculate the route to use at each hop by calculating the shortest path based on metrics associated to each link. To offer QoS guarantees to the different classes of service, a QoS-aware routing is the best fit to find a path meeting the constraints imposed by the demands. The associated metrics are generally made dependent on the flow traversing the link, so that the load conditions in the network are taken into account when calculating the best route for every pair of nodes (and indirectly determining a *working layout*). However, as the metrics are dynamically updated to reflect the load conditions in the network, route oscillations will be produced. Load sharing techniques could help in reducing the probability of the route oscillations, but in a pure IP environment, the use of load sharing is limited to equal costs paths to avoid routing cycles.

MPLS can help introduce the required TE functionalities, while avoiding the cited problems of route oscillation and route cycles. The fact that all packets labelled with the same FEC are guaranteed to be delivered over the same route allows us to define a *virtual* topology constituted by the LSPs mapped onto the physical topology. There are two ways in which the optimal virtual layout can be calculated: *on-line* and *off-line*.

On-Line Layout Calculation

The idea behind on-line methods is to find a path meeting the constraints upon demand arrivals. The LSPs can be calculated to provide the required resources and to meet the service constraints, while optimizing a particular objective function of the network performance. The resulting LSP can then be established using protocol extensions like RSVP-TE [10] or protocols intended for that purpose like CR-LDP [12].

The problem of finding an optimal path meeting a set of constraints is referenced as *constrained-based routing*. Constrained-based routing constitutes an important area of research. The Minimum Interference routing (MIRA) algorithm [15] tries to find a path avoiding the critically loaded paths while verifying the constraints. [25] improves upon MIRA by choosing a set of k minimum-interference paths meeting the constraints. Generally used costs are related to the objective of minimizing congestion.

Selecting the paths on-line, the long term traffic distribution in the network depends on the order and size of demand arrivals. As a consequence, the resulting layout will not be globally optimal with respect to a global performance objective. In that case the point of operation cannot be fixed, as the paths are established as needed, and calculated using local information on the state of the network.

Off-Line Layout Calculation

In the context of the presented TE closed loop system, we are interested in fixing an operational point. As such, we need to resort to an off-line calculation of the virtual layout. The off-line calculation has the advantage of allowing the use of global information, so for a given traffic demand, a globally optimal layout can be calculated with respect to a cost function representing the OAM costs. Cost functions generally used consider transport costs or traditional performance measures like total delay. In the present work, realistic cost functions are proposed representing the costs as seen by the operator: operation and maintenance costs. The OAM costs in large networks are related to the complexity (i.e. number of hops and paths) needed to route a given demand. A suitable objective will be then to minimize the complexity of the virtual layout being calculated. Each path has an associated cost if the path is being used, representing the particular cost structure of the operator. Given that QoS guarantees must be ensured, such as the end-to-end path delay and throughput, the calculation includes these guarantees as constraints.

The problem of minimizing the total number of paths in a layout has been already addresses in the field of optic network design and planning. The results obtained for optic networks are useful as a reference, but they cannot be directly applied to the design of MPLS layouts because of the different granularities required for the IP transport layer. At the same time, the large bandwidths available in a *lightpath* allow for the end-to-end path delay constraints to be expressed simply as a limit in the number of *hops* used in a given path, while in a MPLS context these constraints are modelled in a more realistic way given the smaller bandwidths required for each path.

A first contribution in the present work constitutes the formulation of the layout design problem minimizing the complexity required, while meeting realistic QoS constraints imposed by the demands.

3.3.2. Varying Traffic Conditions: Network Reconfiguration

When the traffic dynamics is considered, the point of operation is not the optimal anymore, as the traffic matrix for which the current operational layout has been calculated doesn't represent the current demands. A given operational point can *absorb* more or less important

traffic variations thanks to the TE mechanisms used inside shorter terms control loops, but for significative changes in demand conditions, calculating a new operational point becomes unavoidable.

The new layout has to also take into account the objectives introduced in section 3.3.1. However, when the number of established paths in the current layout is large, the transition between two optimal layouts in the sense of the presented objective can be very costly. Indeed, if all paths of the new layout have to be established in parallel in order to minimize service disruption times, the operator will have to make enough resources available for this to happen. On the other hand, if all the paths in the new layout are set up once the paths in the old layout have been released in order to minimize the need for spare network resources, then the service disruption time may be increased to levels incompatible with the service contracts. We identify the objective of minimizing the transition complexity, together with the complexity of the new calculated layout to be an important contribution in the field of MPLS layout design considering the traffic dynamics.

4. Contribution to the Dimensioning of MPLS Networks: Design of Reduced Complexity Layouts

4.1. Motivations and Previous Work

Assuming that the network topology has already been deployed to optimize the cost functions considered in the planning phase, which includes the capacity planning and node placement problems [42, 43, 44], the operator faces the problem of how to design, for a given traffic demand, the *virtual* topology (i.e. the set of LSPs) mapped onto the physical topology. Indeed, evolved traffic engineering on source routed paths allows to split the load of the total demand for a given node pair among the various LSPs connecting that node pair. In this way, the optimal routing with respect to some measure of operations and maintenance costs can be effectively implemented, by optimizing the way the total demand for all node pairs in the network is routed through this virtual topology.

Both, off-line and on-line approaches have been proposed throughout the literature to design the layout. The idea behind on-line approaches is to find a suitable path upon demand arrivals. LSPs can be calculated to meet demand requirements and service constraints, such as available capacity and experienced delay, as well as to optimize some function of network economics. Calculated paths will be established using CR-LDP [12] or RSVP-TE [10]. In the on-line path calculation approaches, the resulting global traffic distribution over time depends on the arrival order and size of demands. Actual resource allocation over time could become far from optimal with respect to some performance criteria (e.g. maximum link load or total delay) or economic criteria (e.g. quantity of paths or links) used to design the layout. Off-line LSP layout calculation allows for the setting up of a point of operation, globally optimal with respect to some performance criteria related to the network cost of operation. Off-line calculation takes into account global information about the state of the network and traffic forecast, while on-line calculations are only allowed to take into account local and incomplete information. In fact, on-line and off-line calculations are complementary. The network can be optimally engineered and set up around a point of operation on a long term basis, and on-line decisions can be taken to accommodate traffic variations around that point on a shorter timescale.

Usually proposed cost functions for layout optimization consider the transport cost seen by each user imposing a demand [22], minimizing the most loaded link in the network [23], or

minimizing the average cross network packet delay, subject to a delay constraint for each origin-destination pair [24].

In this chapter, we take into account the operation and maintenance cost of the network. In order to do so, we focus on the problem of obtaining a layout which is optimal in the number of required LSPs. Indeed, cost of operation in large networks in directly related to the layout complexity. We use weights associated to each path in order to reflect a particular operator's cost structure. The problem of minimizing the total number of paths has been analyzed in the context of DWDM networks [25]. The related results cannot be applied in our context since the delay constraints cannot be expressed here just as maximum number of hops and thus linear programming approaches are not applicable to solve our problem. We formulate the problem, which we call Minimum Path Set and Flow Allocation Problem (MPSFAP), as a Mixed Integer Non Linear Problem (MINLP). Two different cost functions are proposed. The first one aims at minimizing a function of the total number of paths, under an end-to-end delay constraint. The second one aims at minimizing an increasing function of the total number of paths and of the total delay under an end-to-end delay constraint. The second approach leads to a more homogeneously distributed load. We first consider a unique service class, and give QoS guarantees for it, by imposing a maximum delay constraint on each path. The model is then extended to consider multiple service classes.

4.2. Network Model Notation

For our modeling purposes, the underlying physical network provides transmission lines between connecting nodes. These transmission lines have an associated physical capacity. According to the traffic engineering techniques reviewed in Chapter 3, we identify the nodes with MPLS Label Switching Routers (LSR), originating and terminating Label Switched Paths (LSP) over which the traffic will be transported between source and destination nodes. The LSPs can be established using either manual or automated procedures via a management system through signalling protocols (e.g. CR-LDP or RSVP-TE). The LSPs are established over the transmission lines connecting the nodes, and allocated a part of the available capacity.

Formally, the physical network is represented by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices indexed $1, 2, \ldots, N$; $\mathcal{E}$ is the set of directed edges indexed $1, 2, \ldots, M$, corresponding to the links in the network. Link i, corresponding to a transmission line, has capacity C_i. Let $\mathbf{C}$ be the single column matrix representing the M-dimensional capacity vector.

Paths

The virtual topology mapped onto the physical topology allows for evolved traffic engineering to be implemented on the network, making a more efficient traffic distribution possible. This efficiency stems from the fact that the traffic can be splitted among different paths instead along a unique path as is the case with interior gateway protocols (IGP) in a pure IP environment. Let's assume that every pair of nodes in the physical topology is connected through one or more paths, corresponding to the different LSPs in the layout. We use the terms *path* and *LSP* as synonymous.

Let the node pairs (m, n), with $m \neq n$, be indexed $1, 2, \ldots, Q$ where $Q = N(N-1)$ is the total quantity of node pairs. For each source-destination pair $q = (m, n)$, we identify a *commodity* with the demand requested for q as d_q. The commodity d_q may be routed via the K_q different acyclic paths (or routes) binding node m to node n over the graph. These paths are indexed $a_q^1, a_q^2, \ldots, a_q^{K_q}$, and are known in advance. Any path a_q^k can be represented as a vector with entries $a_{q,i}^k = 1$ if path a_q^k uses link i, and 0 otherwise, for $i = 1, 2, \ldots, M$. As stated, we only consider acyclic paths, i.e. no path traverses twice the same link or the same node. We can represent the $K = \sum_{q=1}^{Q} K_q$ single paths between all pair of nodes as the $M \times K$ arc-path incidence matrix $\mathbf{A}$:

$$\mathbf{A} = \left(\begin{array}{ccc|ccc|c|ccc} a_{1,1}^1 & \ldots & a_{1,1}^{K_1} & a_{2,1}^1 & \ldots & a_{2,1}^{K_2} & \ldots & a_{Q,1}^1 & \ldots & a_{Q,1}^{K_Q} \\ a_{1,2}^1 & \ldots & a_{1,2}^{K_1} & a_{2,2}^1 & \ldots & a_{2,2}^{K_2} & \ldots & a_{Q,2}^1 & \ldots & a_{Q,2}^{K_Q} \\ \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ a_{1,M}^1 & \ldots & a_{1,M}^{K_1} & a_{2,M}^1 & \ldots & a_{2,M}^{K_2} & \ldots & a_{Q,M}^1 & \ldots & a_{Q,M}^{K_Q} \end{array}\right)$$

The matrix $\mathbf{A}$ can be represented synthetically as the block matrix:

$$\mathbf{A} = \left(A_1 \,\middle|\, A_2 \,\middle|\, \ldots \,\middle|\, A_Q \right) \tag{4.1}$$

where the block A_q is the arc-path matrix corresponding to the K_q paths for the node pair q.

Let $\mathbf{W}$ be the single column matrix representing the K-dimensional vector containing the weight w_q^k associated with path a_q^k. In section 4.3 we present a way weights can be used to fine tune the path cost structure for a given operator.

Demands

In order to characterize the demands to every pair of nodes, we need to understand the nature of the IP traffic. We can analyze it at different timescales: packet and flows.

Packet Level Characteristics

It is well known that the stochastic processes describing the arrival of IP packets exhibit a high variability at all timescales [27]. Consequently, the use of a token bucket to describe its behavior is not well adapted. The description of IP traffic between any pair of nodes should be made therefore in terms of flows or aggregates.

Flow Level Characteristics

A flow can be defined as a succession of packets near in time to one another, generated by the same application instance between a particular source and destination pair. We can identify two great categories of flow types [20]: elastic and stream flows. Stream flows are representative of real-time applications (e.g. audio or video); they have intrinsic temporal properties that the network must preserve. The elastic flows are representative of file transfer-like applications. For elastic flows the transfer throughput depends on the available bandwidth: the transfer duration is a function of the file size and traffic characteristics of the transport network during the transfer. Elastic traffic is the most important of the Internet in terms of its proportion to stream traffic (approximately 80% of the flows). To describe the flow arrival process, the usual practice is to characterize it as a Poisson process. For the stream flow types, we can assimilate it to the arrival of voice calls, and the Poisson description would be appropriate. For the elastic flows, a detailed study of the behavior of a typical user during a session (e.g a web session) should be considered. However, empirical results show that we can appropriately assimilate the arrival process of elastic flows to a Poisson process in a given link at the core of the network [28].

Demand Characterization

Measures on Internet backbone links have shown that the traffic on those links is more or less predictable [29]. The traffic intensity presents cycles, resulting on long high utilization periods, followed by long low utilization periods during the day on weekdays. The same periodicity can be found during the weekend, but with different intensities and times of the day. This periodicity suggests that the traffic intensity could be modelled using a stationary stochastic process. More precise studies regarding the modelling of traffic processes at flow level can be found in [85, 86, 87]. The average measured traffic could be then interpreted as a demand with a throughput equal to the product of the flow arriving rate and the average throughput of the individual flows (both stream and elastic).

The use of flows allows us to better characterize the demands, and as such, to also characterize the QoS offered by the network. Based on these models, we can calculate the required dimensioning to provide the required QoS guarantees. The flow level corresponds to the timescale at which we apply the traffic engineering mechanisms to control the offered QoS (i.e. short timescale corresponds to packet level and longer timescales to flow level). In conclusion, we propose a flow level description of demands through the single column matrix:

$$\mathbf{D} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_Q \end{pmatrix} \tag{4.2}$$

with entry d_q being the traffic value requested for the node pair q, for $1 \leq q \leq Q$

Path Flows

The optimal dimensioning of the network with respect to an objective function given by the operator would make use of multiple paths connecting every pair of nodes. Each path in the set of paths between a particular source and destination is allocated a proportion of the demand for that node pair. The proportion of flow allocated to each path is also a result of the layout optimization. Figure 4.1 shows the path flows and their relationship to the demands and the link capacities. We can easily see that the total amount of flow allocated to the set of paths connecting a given pair of nodes must equal the demand for that pair of nodes; this equality gives rise to the *demand constrain* in the mathematical problem formulation. In the same way, we can see that a given link is traversed by paths connecting different pair of nodes, so the sum of flow allocated to each of such paths must be less or equal the capacity of that link; this inequality gives rise to the *capacity constraint* in the mathematical problem formulation.

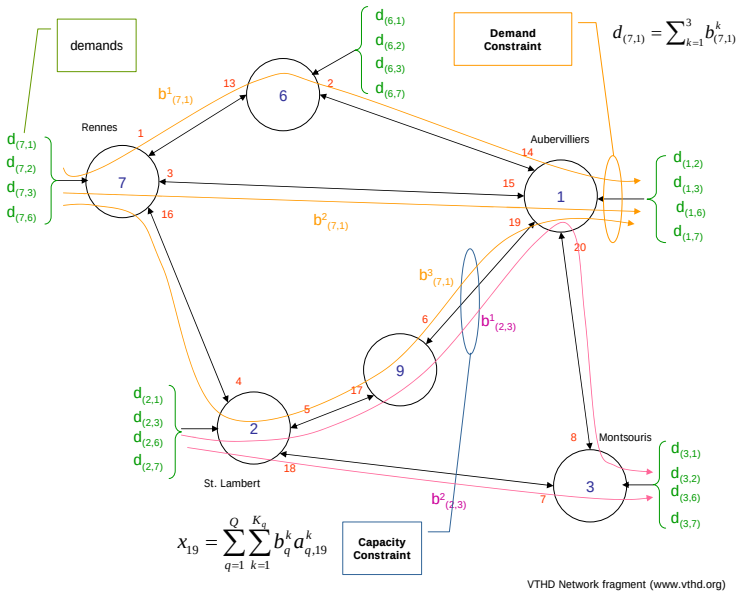


Figure 4.1: Path Flows: Relationship to Demands and Link Capacities

Formally, let the flow assigned to path a_q^k be denoted by b_q^k. The single column matrix representing the path flows is:

$$\mathbf{B} = \begin{pmatrix} b_1^1 \\ \vdots \\ b_1^{K_1} \\ \hline \vdots \\ \hline b_Q^1 \\ \vdots \\ b_Q^{K_Q} \end{pmatrix} \tag{4.3}$$

Let us define $\mathbf{R} = (r_{q,i})$ the $Q \times K$ commodity-path incidence matrix, where entry $r_{q,i} = 1$ if i lies in the interval $[K_1 + K_2 + \ldots + K_{q-1} + 1, K_1 + K_2 + \ldots + K_{q-1} + K_q]$ and 0 otherwise, for $1 \leq q \leq Q$ and $1 \leq i \leq K$. We assume no flow loss, hence the sum of the flows for the K_q paths connecting the node pair q must satisfy the demand d_q, for all $1 \leq q \leq Q$:

$$\mathbf{R} \cdot \mathbf{B} = \mathbf{D} \tag{4.4}$$

Let x_i be the value of the total flow traversing link i, for $1 \leq i \leq M$, and let $\mathbf{X}$ be the single column matrix representing the M dimensional vector of total flows for all links:

$$\mathbf{X} = \mathbf{A} \cdot \mathbf{B} \tag{4.5}$$

4.3. Building the Cost Functions

The cost of operation for large networks is related to the layout complexity [16]. We define complexity as the number of paths necessary to transport a given demand matrix $\mathbf{D}$. Indeed, the operations and maintenance costs are related to the quantity of paths to establish when setting up the layout and then maintained once the layout is in place. Traditionally, objectives for layout optimization consider the minimization of some measure of network performance. Functions of total cross network delay or link load [22, 23, 24] are generally proposed as objectives for layout optimization. Layouts obtained with these objectives will distribute the flows on the paths so as to equalize the loads on the links, resulting in a low delay variation from link to link. Models proposing those objective functions assume a cost proportional to the use of the capacity, which is not realistic in terms of operations and management costs from an operator's standpoint. As we discussed in previous chapters, the cost of capacity deployment is already considered in the capacity planning phase. We are focusing on the dimensioning of the network (i.e. the obtention of a layout and flow allocations) given the physical topology.

To minimize OAM costs we need to take into account some function of layout complexity, provided that the QoS guarantees given to the customers are always met. We introduce a way to provide these QoS guarantees through constraints in the optimization problem in section 4.4. The objective will be then to minimize a function of the general form:

$$\text{Minimize} \left[\sum_{q=1}^{Q} \sum_{k=1}^{K_q} w_q^k h_q^k \right] \tag{4.6}$$

where h_k^q is a binary variable indicating if the corresponding path a_q^k is in use or not. We define $\mathbf{H}$ as a single column matrix, whose entries h_q^k are defined as follows:

$$h_q^k = \begin{cases} 1 & \text{if } b_q^k > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

The objective function of the form given in (4.6) will minimize the weighted sum of the binary variables. The weight associated to each path represents then the cost that the set-up of such path has for the operator. The way in which such weights are defined can be used by the operator to fine tune the cost structure of the network OAM.

If the path weights are independent of the path flow, the layouts optimizing the objective function (4.6) will tend to concentrate the flow on fewer paths. One of the consequences of this result is that experienced end-to-end delay along the different paths connecting any given pair of nodes will exhibit a great dispersion from the average for that group of paths. If a packet-based routing strategy is to be used, this dispersion will produce a high rate of resequencing at the end nodes, since IP packets using paths with different end-to-end delay will not arrive in order to the destination. If a flow-based routing strategy is to be used, the need for resequencing is minimized, since packets belonging to the same flow will traverse the same path. In order to minimize the dispersion in the end-to-end path delay on the layout, we propose an alternative objective function, which makes appear a second term taking into account the total cross network delay:

$$\text{Minimize} \left[\alpha \sum_{q=1}^{Q} \sum_{k=1}^{K_Q} w_q^k h_q^k + \beta \sum_{q=1}^{Q} \sum_{k=1}^{K_q} h_q^k \sum_{i=1}^{M} \frac{\lambda a_{q,i}^k}{C_i - x_i} \right] \tag{4.8}$$

Assuming that it is appropriate in our model to approximate the link delay[1] by an M/M/1 queueing system [30], the second term in 4.8 will produce layouts which tend to equalize the load distribution through the network, while keeping the complexity low. The factors α

[1] Neglecting packetization and transmission delays

and β help us in controlling the relative importance of each term in the objective function, and λ is the average packet size. Indeed, as we expressed in section 4.2, we are assuming that a Poisson process is a good model for the flow arrival process. At the same time, as the dimensioning refers to the network core, the flows are seen at an aggregate level. The hypotheses to apply the Kleinrock independence approximation for each link delay then hold[2].

Path Weights

The weights w_q^k associated with each path can be used by the operator to fine tune the cost structure of the path layout. For instance, setting the weights on a hop-count basis, the operator can introduce the fact that setting up shorter paths is less expensive for its management and maintenance cost structure than setting up paths with longer hop-counts:

$$w_q^k = \sum_{i=1}^{M} a_{q,i}^k \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.9}$$

In particular, hop-count based weights minimize a function of the number of used links. Setting all the weights to 1 minimizes the total number of paths. Operators having different cost structures might use these weights to represent their own operational costs. In what follows, we use (4.9) to calculate the weights associated to each path. As results show, minimizing the total number of links, indirectly minimizes the total number of used paths, and we assure at the same time that shortest paths (in terms of hops) are taken first.

4.4. Setting QoS Guarantees

If no performance measures are included in the objective function, we need a way to ensure that QoS guarantees are met. These guarantees will then be introduced in the problem formulation as constraints. Those constraints ensure that no path will exceed a flow allocation on component links that violates the required QoS objectives. We are interested in providing two types of QoS guarantees: throughput and end-to-end delay. The demand constraints ensure that enough bandwidth is allocated to the set of paths connecting any given pair of nodes, so as to guarantee the throughput for that node pair. Identifying a commodity with all the demand for the node pair q, the bandwidth guarantee is given for the total demand d_q:

$$d_q = \sum_{k=1}^{K_q} b_q^k \tag{4.10}$$

[2] Assuming packet sizes exponentially distributed, a densely connected network and moderate to heavy load

Paths in this context will then be identified as transporting an *aggregate* of traffic, and guarantees are given for that aggregate. A discussion about different alternatives on guarantees for multiple classes of service (both absolute and relative) is given in section 4.7.

Guarantees for delay are given through the end-to-end path delay constraints. Any given path in the set of paths connecting a given node pair q, which has an allocated path flow $b_q^k > 0$, cannot exceed a given end-to-end maximum allowed delay θ_q:

$$h_q^k \sum_{i=1}^{M} \frac{\lambda a_{q,i}^k}{C_i - x_i} \leq \theta_q \tag{4.11}$$

for all the paths in the set $k = 1, \ldots, K_q$ connecting the node pair q. Again, this guarantee is given for any packet of the aggregate d_q traversing any path connecting the node pair q. A discussion about separate guarantees given for different classes of service on the aggregate d_q is given in section 4.7.

It is important to note here that the end-to-end path delay constraint should be applied to *used* paths only (i.e. paths with a positive flow allocation). Indeed, the path delay for a given commodity not only depends on the value of its own traffic, but also on the traffic issued by all the other paths sharing the same links, as can be seen in an example depicted in Figure 4.2, and Table 4.1.

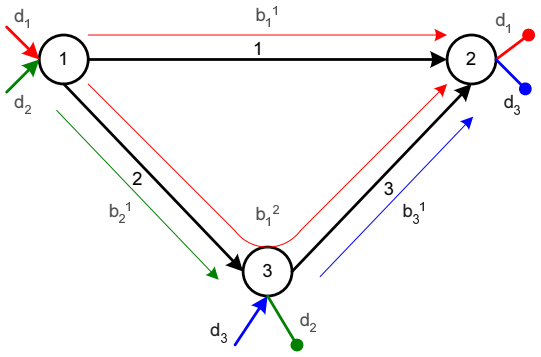


Figure 4.2: Example on the importance of avoiding path-delay constraints on unused paths

In the referred example, packets sent along the path a_1^2 would see a total delay of 18.18 msec., which is greater than the maximum allowed delay $\theta_1 = 10$ msec. This violated constraint will prevent paths a_2^1 and a_3^1 from being able to carry the corresponding flows to meet the demands. If no path delay constraint is imposed on unused paths, both a_2^1 and a_3^1 will be able to carry the flows b_2^1 and b_3^1 to meet the demands without violating any constraint.

commodities

q	(s,d)	d_q(bps)	θ_q(msec)
1	$(1,2)$	890	10
2	$(1,3)$	890	10
3	$(3,2)$	890	10

links

link	C_i(bps)	x_i(bps)
1	1000	890
2	1000	890
3	1000	890

paths

a_q^k	links	b_q^k(bps)	τ_q^k(msec)
a_1^1	1	890	9.09
a_1^2	2, 3	0	18.18
a_2^1	2	890	9.09
a_3^1	3	890	9.09

τ_q^k is the calculated end-to-end path delay for path a_q^k

Table 4.1: Example on the importance of avoiding path-delay constraints on unused paths

4.5. Formulation: Minimum Path Set and Flow Allocation Problem (MPSFAP)

Given the network physical topology and the traffic matrix, the problem is to find a set of paths and the flow allocation over those paths in order to optimize a cost objective, complying to some QoS constraints. In this section we formulate the problem of designing the optimal layout w.r.t. the complexity of the layout (as defined in section 4.3, while meeting some QoS guarantees (i.e. end-to-end path delay and available bandwidth per node pair). The *Network Design Multicommodity Flow Allocation Problem* in its matrix form is stated as follows:

MPSFAP 1
Minimum Path Set and Flow Allocation Problem

Given:

$$\mathbf{A}, \mathbf{C}, \mathbf{D}, \mathbf{\Theta}, \mathbf{\Delta}, \mathbf{W}$$

minimize :

$$^t\mathbf{W} \cdot \mathbf{H} \tag{4.12}$$

subject to:

$$\mathbf{X} = \mathbf{A} \cdot \mathbf{B} \leq \mathbf{C} \tag{4.13}$$

$$\mathbf{R} \cdot \mathbf{B} = \mathbf{D} \tag{4.14}$$

$$h_q^k \sum_{i=1}^{M} \frac{\lambda a_{q,i}^k}{C_i - x_i} \leq \theta_q \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.15}$$

$$b_q^k \leq h_q^k \delta_q \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.16}$$

$$h_q^k \in \{0, 1\} \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.17}$$

$$b_q^k \geq 0 \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.18}$$

where $\mathbf{\Theta}$ is the column matrix containing the maximum tolerable delay θ_q for each commodity d_q and $\mathbf{\Delta}$ is the column matrix containing the maximum flow δ_q acceptable on one path for the commodity d_q.

Similarly, we can define a problem MPSFAP 2 using the objective function (4.8) to take care of the flow distribution in the layout:

MPSFAP 2
Minimum Path Set and Flow Allocation Problem Considering Flow Distribution

Given:

$$\mathbf{A}, \mathbf{C}, \mathbf{D}, \mathbf{\Theta}, \mathbf{\Delta}, \mathbf{W}$$

minimize :

$$\alpha \quad {}^t\mathbf{W} \cdot \mathbf{H} + \beta \sum_{q=1}^{Q} \sum_{k=1}^{K_q} h_q^k \sum_{i=1}^{M} \frac{\lambda a_{q,i}^k}{C_i - x_i} \tag{4.19}$$

subject to:

$$\mathbf{X} = \mathbf{A} \cdot \mathbf{B} \leq \mathbf{C} \tag{4.20}$$

$$\mathbf{R} \cdot \mathbf{B} = \mathbf{D} \tag{4.21}$$

$$h_q^k \sum_{i=1}^{M} \frac{\lambda a_{q,i}^k}{C_i - x_i} \leq \theta_q \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.22}$$

$$b_q^k \leq h_q^k \delta_q \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.23}$$

$$h_q^k \in \{0, 1\} \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.24}$$

$$b_q^k \geq 0 \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.25}$$

Constraints (4.13)/(4.20) and (4.15)/(4.22), both impose limitations in the usable capacity on each link. However, the constraints (4.15)/(4.22) are more restrictive in the sense that solutions meeting these constraints, will also meet the constraints (4.13)/(4.20) for all links. We keep all constraints in our formulation for clarity, but they will be automatically simplified in the *presolve* phase of the solvers. The additional constraints (4.16)/(4.23) are useful to prevent paths for a given commodity to take all the available capacity of a particular link (e.g. if we want to make possible to share a critical link among paths connecting different node pairs and provide bandwidth guarantees for each path), or to somewhat control *a priori* the load sharing granularity when designing the layout (i.e. we know that each path for commodity d_q will not transport more than δ_q units of bandwidth).

In section 4.6 we compare results obtained for both MPSFAP 1 and MPSFAP 2 for small

networks.

4.6. Preliminary Results on MPSFAP and Model Validation

The MPSAFP problems are *network design multicommodity flow problems*, since the binary variables h_q^k make decisions on whether a path is to be included in the solution or not, and the continuous flow variables b_q^k indicate the quantity of flow to be allocated to each of the selected paths. The MPSFAP 1 problem can be seen as solving two related problems at the same time: one *0-1 network design problem* as only the binary variables are present in the objective function, and a *constraint satisfaction problem* on the path flow variables. The set of paths is given through the arc-path incidence matrix $\mathbf{A}$. The commodity-path incidence matrix $\mathbf{R}$ indicates which of those paths connect what node pairs. The resulting MPSFAP are *Mixed Integer Non-Linear Programming* (MINLP) problems, since they have a combination of integer (h_q^k) and continuous (b_q^k) variables, and a set of linear and non-linear constraints.

Constrained global optimization is $\mathcal{NP}$*-complete*, because it takes exponential time to verify whether a feasible solution is optimal or not for a general constrained NLP [32, 33]. Besides, the MPSFAP problems lie in the category of general MINLP problems, which are $\mathcal{NP}$*-complete* [34, 35]. Even if we formulate the problem using piecewise linear constraints for the path delay instead of the original non-linear constraints, the resulting MILP problem is known to be also $\mathcal{NP}$*-complete*. Replacing the non-linear constraints however is not practical in the context of the design of MPLS layouts, since we need to make no assumptions on the underlying transport technologies, whenever they provide links with an associated available capacity. The underlying transport infrastructure will then provide links with different capacities, leading to a different piecewise linear function for each possible capacity value. This adds to the size of an already large and complex problem, relativizing the gains obtained by formulating a MILP problem instead of a MINLP problem. We resort then to heuristics to efficiently solve the MPSFAP problems in Chapter 5.

The MPSFAP problems are particularly difficult to solve due to their size and complexity. We use a *path formulation* instead of a *flow formulation* in order to reduce the size of the problem [14], but we pay the price of having an enormous set of variables. The MPSFAP problems have two variables for each path: one integer variable representing the decision variable of the network design problem, and one continuous variable representing the amount of flow to allocate to that particular path in the constraint satisfaction problem (in particular for the MPSFAP 1 problem, which we consider the most interesting in terms of practicality as we can safely assume flow-by-flow routing in the network core). The quantity of variables is $2K$, where K is the total number of paths connecting every

pair of nodes (i.e. the number of rows in the arc-incidence matrix). The number of all possible paths will typically be enormous, growing exponentially with the network size [14]. MPSFAP problems have Q demand constraints, M capacity constraints, and K path delay constraints, in addition to the nonnegativity conditions imposed on the path flow values.

Due to the limitations imposed by the size of the problems to solve, we first obtain preliminary results from a solver for small networks. The objective is to validate the model and obtain a first insight on the results obtained in view of the heuristics to develop in the next chapter.

4.6.1. Traffic Matrixes

Traffic demands are randomly generated according to the following method, which is inspired in the method described in [25]. A certain proportion p of the node-pairs are assigned a demand flow following a random variable uniformly distributed within the interval $[0, \frac{C_q}{a}]$, where C_q is the total capacity available for connecting the node pair q if no traffic is flowing through the network, and a is an arbitrary integer to help us in *scaling* the traffic intensity when all node pairs are considered. Indeed, increasing a would produce traffic matrixes with larger probability of being feasible solutions for a given network topology. To model some degree of asymmetry in the network, we allow a proportion $(1-p)$ of the node pairs to have traffic demands uniformly distributed in the interval $[0, \frac{C_q Y}{a}]$. We define Y as the ratio $\frac{max_q\{C_q\}}{min_q\{C_q\}}$, which are the capacity of the best connected node pairs and the capacity of the worst connected node pairs in the network respectively.

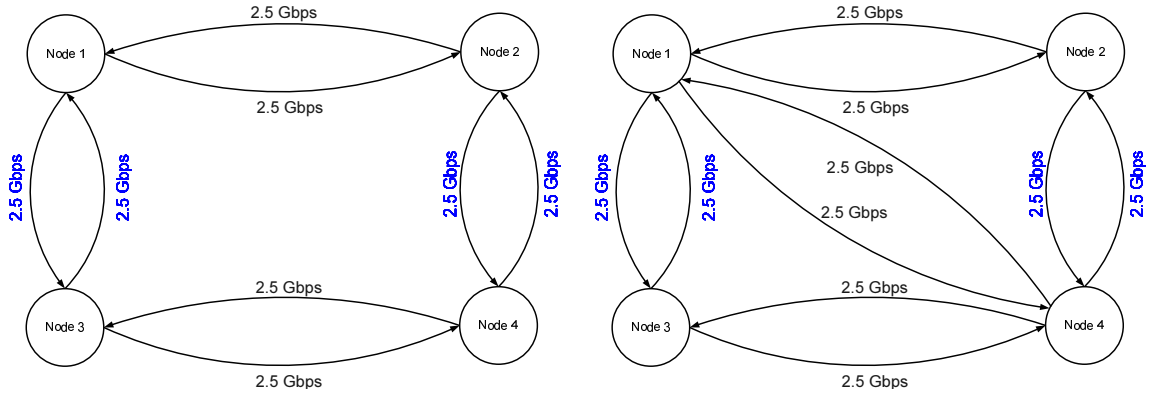
A set of 25 traffic matrixes are randomly generated for each of the test topologies described in section 4.6.2.

4.6.2. Network Topologies

In order to study the problem and validate the model we resort to two small sized network topologies, so that the size of the problem is tractable by known solvers in the domain. The network topologies proposed are a 4-node network with 8 and 10 (directed) links respectively. We denote these test topologies by NET1 and NET2. Figure 4.3 shows the corresponding network topologies.

Link capacities for both networks are set to 2.5 Gbps for all links. NET1 has a total of 24 possible paths, generating a problem of 48 variables and 68 constraints (besides the path flow positivity conditions and the simplifications in the *presolve* phase of the solver). Similarly, NET2 has a total of 38 possible paths, generating a problem of 76 variables and

98 constraints. These problem sizes allow us to obtain preliminary results useful to evaluate the interest of the proposed cost functions.



(a) NET1 (b) NET2

Figure 4.3: NET1 and NET2 Networks.

4.6.3. Reference Problem

One of the most used cost functions through the optimization literature is a measure of network performance such as total delay in the network. We are interested in evaluating the complexity of the layouts created with our proposed cost functions to the complexity of layouts obtained with a cost function minimizing total delay in the network. We propose then a *reference problem* which minimizes total cross network delay, and we will denote this problem by *Minimum Total Delay Flow Allocation Problem* (MTDFAP). The proposed objective for MTDFAP is to minimize:

$$\sum_{q=1}^{Q} \sum_{k=1}^{K_q} h_q^k \sum_{i=1}^{M} \frac{\lambda a_{q,i}^k}{C_i - x_i} \tag{4.26}$$

for all pairs $q = 1, \ldots, Q$, subject to the same constraints as defined for the MPSFAP problems (4.13)(4.15)(4.14) for MPSFAP1, and (4.20)(4.22)(4.21) for MPSFAP2. MTDFAP is also formulated as a MINLP problem. Unfortunately, the MTDFAP problem can not be used to compare our results for larger networks using the proposed heuristics in Chapter 6 because of the size of the problem instances generated.

4.6.4. Interface to Solvers: Modeling Language

As a first approach to solving the MPSFAP problems, numeric solvers can help to get an insight in the problem structure and complexity. A numeric solver can help also in validat-

ing the problem formulations through the use of a formal mathematical modeling language extensively used in the field of operations research. We have searched first for a suitable solver, since they are generally adapted to a very narrow type of problem and particular structures of the associated matrixes. We resorted to a well-known source of solver implementations, which makes available a set of solvers, categorized by problem type and input interface: the NEOS server on the Internet [18, 37, 38].

The MINLP solver [19] available through NEOS server provides the solution of mixed integer nonlinearly constrained optimization problems in AMPL format. MINLP is suitable for large nonlinearly constrained problems with a modest number of degrees of freedom. MINLP implements a branch-and-bound algorithm searching a tree whose nodes correspond to continuous nonlinearly constrained optimization problems. The continuous problems are solved using filterSQP [39], a Sequential Quadratic Programming solver which is suitable for solving large nonlinearly constrained problems.

AMPL is a *mathematical modeling language*, which provides a standardized way of representing or *modeling* a problem. If we wanted to present a solver with a particular *instance* of a problem, we would have to change the equation system (right hand side terms, quantity of equations, coefficients in equations, etc.) according to each instance. If we used a modeling language instead, we would present the solver with a *model* of the problem, which describes its structure, and then the solver would be capable of *instantiate* every problem according to the *data* associated. There are two common modeling languages widely used in the operational research world: A Modeling Language for Mathematical Programming (AMPL) [36], and General Algebraic Modeling System (GAMS) [84]. Solvers are increasingly being developed to accept one or both of the input formats. The solver must then implement an AMPL or GAMS compilator to create the problem instances with the associated data presented at its input. Once the instance created (correctness of the model and data is checked during compilation), the solver executes a *presolve* phase to detect whether the problem can be simplified (constraints and variables eliminated or transformed). The *simplified* problem is then passed to the solver.

4.6.5. Results and Analysis

Solutions for NET1 and NET2 for 25 traffic matrixes generated according to the model in section 4.6.1 are obtained from the exact (deterministic) MINLP solver available through the NEOS server. Solution for MPSFAP1 and MPSFAP2 problems are compared to the solutions obtained for the MTDFAP problem using the same parameters and topologies. Both MPSFAP1 and MPSFAP2 problems have associated path weights w_q^k set to the number of hops used by the path. Parameters are set to $\delta_q = 2.5$ Gbps. (i.e. the path flows can

use up to the maximum available capacity in any link), end-to-end path delay limits θ_q are set to 30 μsec. The factors α and β are set to 1 for the MPSFAP2 problem. The set of 25 traffic matrixes are generated with parameters $a = 3$ and $a = 4$ for the NET1 network; and $a = 5$ and $a = 6$ for NET2 network. In all cases, we use $p = 0.2$, meaning that 20% of node pairs are allowed demands up to $Y = 1$ (there are exactly 2 possible paths connecting every pair of nodes in NET1) times larger for NET1 and $Y = 1.33$ (there are some node pairs connected through 2 paths and some connected through 3 paths in NET2) times larger for NET2.

NET1 - Hops and Paths												
Matrix	$a = 4$ / $p = 0.2$						$a = 3$ / $p = 0.2$					
	MTDFAP		MPSFAP1		MPSFAP2		MTDFAP		MPSFAP1		MPSFAP2	
	Paths	*Hops*	*Paths*	*Hops*	*Paths*	*Hops*	*Paths*	*Hops*	*Paths*	*Hops*	*Paths*	*Hops*
1	12	16	12	16	12	16	14	20	12	16	12	16
2	12	16	12	16	12	16	12	16	12	16	12	16
3	12	16	12	16	12	16	13	18	12	16	12	16
4	13	18	12	16	12	16	I	I	I	I	I	I
5	12	16	12	16	12	16	12	16	12	16	12	16
6	13	18	12	16	12	16	I	I	I	I	I	I
7	12	16	12	16	12	16	13	18	13	18	13	18
8	13	18	12	16	12	16	13	18	13	18	13	18
9	13	18	12	16	12	16	14	20	14	20	14	20
10	12	16	12	16	12	16	13	18	13	18	13	18
11	12	16	12	16	12	16	13	18	12	16	12	16
12	12	16	12	16	12	16	14	20	13	18	13	18
13	12	16	12	16	12	16	13	18	12	16	12	16
14	12	16	12	16	12	16	12	16	12	16	12	16
15	12	16	12	16	12	16	12	16	12	16	12	16
16	12	16	12	16	12	16	I	I	I	I	I	I
17	12	16	12	16	12	16	12	16	12	16	12	16
18	12	16	12	16	12	16	13	18	12	16	12	16
19	13	18	12	16	12	16	13	18	12	18	13	18
20	12	16	12	16	12	16	13	18	12	16	12	16
21	12	16	12	16	12	16	I	I	I	I	I	I
22	12	16	12	16	12	16	13	18	12	16	12	16
23	12	18	12	16	12	16	I	I	I	I	I	I
24	12	16	12	16	12	16	12	16	12	16	12	16
25	12	16	12	16	12	16	13	18	12	16	12	16
Averages	**12.20**	**16.40**	**12.00**	**16.00**	**12.00**	**16.00**	**12.85**	**17.70**	**12.30**	**16.70**	**12.35**	**16.70**

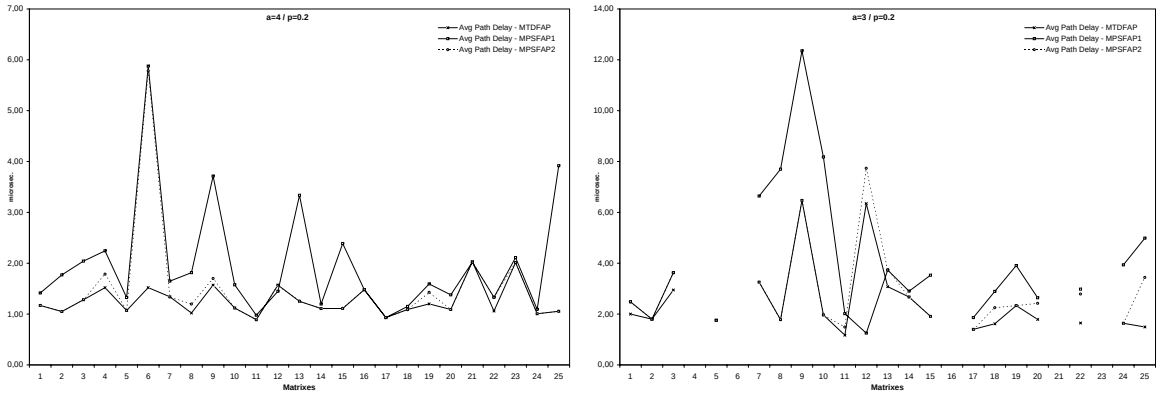
Table 4.2: Quantity of Hops and Paths for MTDFAP, MPSFAP1 and MPSFAP2 Problems with NET1 Network

Table 4.2 and Table 4.3 show the quantity of hops and paths in the solution for the MTDFAP, MPSFAP1 and MPSFAP2 problems for NET1 and NET2 respectively. In both tables, the value **I** indicates an integer infeasible problem as declared by the solver. A set of 25 demand matrixes is generated for each of two load situations, corresponding to mild and heavy load. For NET1 a mild load condition is found with $a = 4$ (all problems feasible), and heavy load with $a = 3$ (20% of the matrixes resulted in infeasible problems). For NET2, a mild load condition is found with $a = 6$ (all problems feasible), and a heavy load is found with $a = 5$ (24% of the problems infeasible). After problem simplification (*presolve*) all the problem instances for NET1 generated a total of 48 linear variables, 24 linear constraints and 44 non-linear constraints, while the problems instances for NET2 generated a total of 76 linear variables, 38 linear constraints and 60 non-linear constraints. The quantity of variables and constraints generated by these simple network topologies give us a taste of

the complexity of the problem larger networks would generate.

The time consumed at the solver server for all problem formulation and all instances of NET1 is under 10 seconds, while the time consumed for NET2 under the same conditions is higher than 100 seconds. We see a time increase of about 10 times when adding a pair of links in the topology (increasing from 24 to 38 possible paths).

Results for NET1 in Table 4.2 show that the number of hops in the layouts resulting from MPSFAP1 and MPSFAP2 are always lower than those in the layouts obtained from MTDFAP. Under a mild load ($a = 4$), MPSFAP1 and MPSFAP2 use in average 2.5% less total hops than MTDFAP. When the load increases ($a = 3$), MPSFAP1 and MPSFAP2 use a $5,99\%$ less total hops than MTDFAP. Looking at the quantity of paths, results show that for mild load MPSFAP1 and MPSFAP2 produce layouts with 1.67% less paths than those produced by MTDFAP, while for heavy load MPSFAP1 and MPSFAP2 produce layouts with around 4.5% less paths than those produced by MTDFAP.

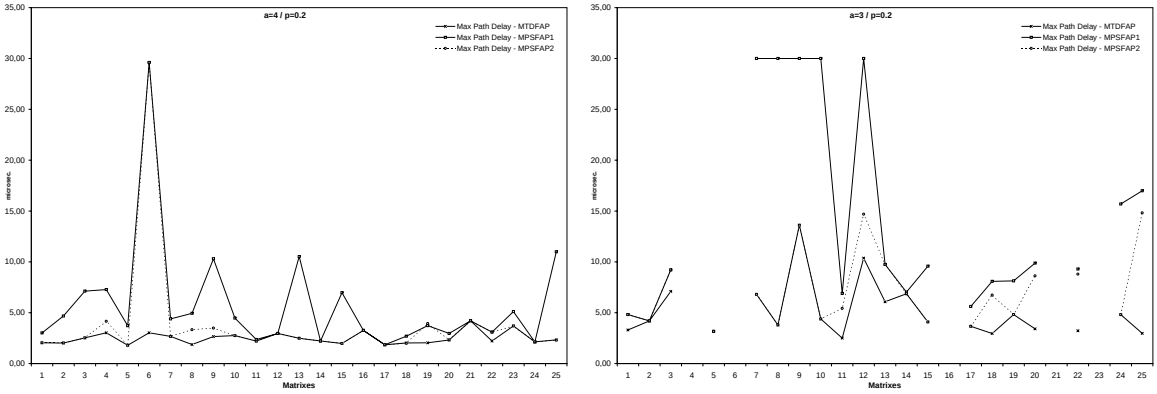


(a) $a = 4$ (b) $a = 3$

Figure 4.4: Average End-to-end Path Delay for NET1

At first sight, both MPSFAP1 and MPSFAP2 produce layouts with the same quantity of hops and paths. Figure 4.4 shows the average end-to-end path delay for mild and heavy load situations. Figure 4.5 shows the maximum end-to-end path delay obtained for every matrix on the set for mild and heavy load situations.

Average end-to-end delay on paths in layouts produced by MPSFAP1 are noticeably higher than those on paths produced by MTDFAP, and the difference increases with the load. Average end-to-end delay on paths in layouts produced by MPSFAP2 are closer to those on paths in layouts produced by MTDFAP. Correspondingly, we see that a higher number of paths reach the maximum allowed end-to-end delay for MPSFAP1 when the load increases, while paths in layouts produced by MPSFAP2 and MTDFAP rarely reach the maximum



(a) $a = 4$ (b) $a = 3$

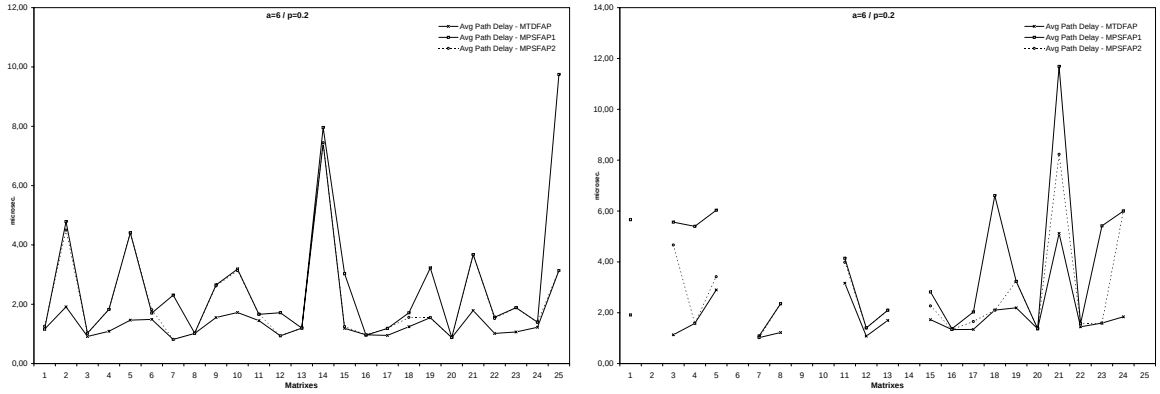
Figure 4.5: Maximum End-to-end Path Delay for NET1

allowed end-to-end delay (*delay constraint*).

NET2 - Hops and Paths												
Matrix	$a = 6$ / $p = 0.2$						$a = 5$ / $p = 0.2$					
	MTDFAP		MPSFAP1		MPSFAP2		MTDFAP		MPSFAP1		MPSFAP2	
	Paths	*Hops*	*Paths*	*Hops*	*Paths*	*Hops*	*Paths*	*Hops*	*Paths*	*Hops*	*Paths*	*Hops*
1	13	16	12	14	12	14	13	16	13	16	13	16
2	14	19	12	15	12	15	I	I	I	I	I	I
3	12	15	12	14	12	14	12	15	12	14	12	14
4	13	16	12	14	12	14	13	16	13	16	13	16
5	13	16	12	14	12	14	14	18	13	16	13	16
6	12	16	12	15	12	15	I	I	I	I	I	I
7	12	14	12	14	12	14	12	14	12	14	12	14
8	12	14	12	14	12	14	13	16	12	14	12	14
9	12	16	12	15	12	15	I	I	I	I	I	I
10	14	18	12	15	12	15	I	I	I	I	I	I
11	12	16	12	15	12	15	13	17	12	16	12	16
12	12	14	12	14	12	14	13	16	12	14	12	14
13	12	14	12	14	12	14	13	17	12	14	12	14
14	13	17	13	16	13	16	I	I	I	I	I	I
15	12	16	12	15	12	15	13	18	12	15	12	15
16	12	14	12	14	12	14	12	14	12	14	12	14
17	12	16	12	14	12	14	12	16	12	15	12	15
18	13	16	12	14	12	14	13	16	13	16	13	16
19	12	15	12	15	12	15	13	18	12	15	12	15
20	12	14	12	14	12	14	12	14	12	14	12	14
21	13	18	12	15	12	15	14	20	14	19	14	19
22	12	16	12	14	12	14	12	16	12	15	12	15
23	13	16	12	15	12	15	13	16	13	16	13	16
24	12	16	12	14	12	14	13	17	12	15	12	15
25	14	18	14	18	14	18	I	I	I	I	I	I
Averages	**12.52**	**15.84**	**12.12**	**14.60**	**12.12**	**14.60**	**12.79**	**16.32**	**12.37**	**15.16**	**12.37**	**15.16**

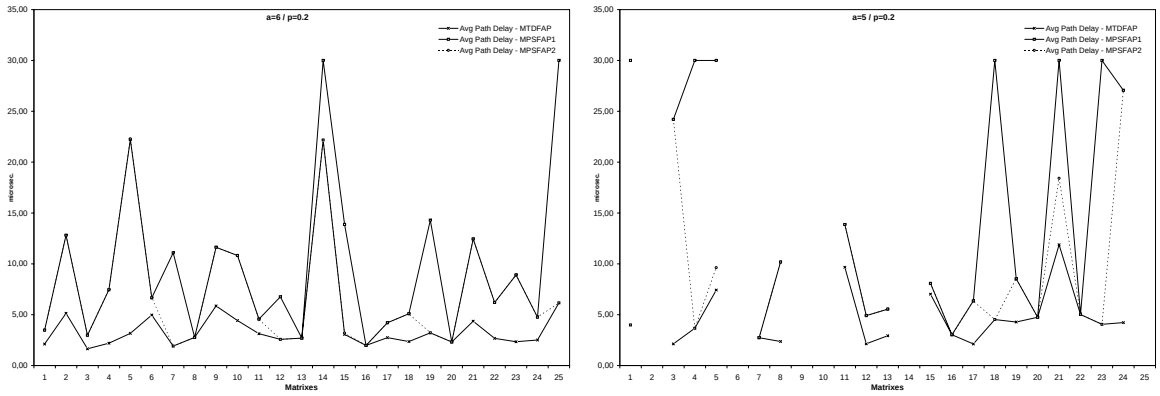
Table 4.3: Quantity of Hops and Paths for MTDFAP, MPSFAP1 and MPSFAP2 Problems with NET2 Network

Results in Table 4.3 show similar trends as results for NET1 in all three problems. In this case, the savings in quantity of hops is of about 8% for MPSFAP1 and MPSFAP2 in all load situations. Similarly, the savings in quantity of paths is of about 3.5% under the same conditions. Figures 4.6 and 4.7 show the average and maximum end-to-end delay for NET2 respectively on paths in layouts obtained with MTDFAP, MPSFAP1 and MPSFAP2 under mild and heavy load conditions.



(a) $a = 6$ (b) $a = 5$

Figure 4.6: Average End-to-end Path Delay for NET2



(a) $a = 6$ (b) $a = 5$

Figure 4.7: Maximum End-to-end Path Delay for NET2

These results suggest that when the network size increases, the savings in quantity of hops and paths can be more significative, encouraging the development of heuristics which would allow us to obtain layouts minimizing complexity for larger networks in order to realize operation and maintenance cost savings. The reason for the increase in savings when network density increases is quite obvious: a denser network presents more paths connecting every pair of nodes. As MTDFAP tries to minimize the total delay in the network, the optimal solution would be to use as much paths as possible, charging each path to a minimum so as to make the delay on each path as low as possible, the limit given by the critical links, which will set the lower bound delays on any path traversing them. Results also show that even when the objective function aims at minimizing the total number of hops in the layout through the choice of the path weights defined in equation (4.9), the number of total paths gets also minimized.

4.7. Extensions for Multiple Classes of Service

In section 4.4 we defined a commodity as an aggregate of traffic to be transported from node m to node n. As such, all packets belonging to the same aggregate are transported through the set of paths connecting the corresponding node pair, all receiving the guarantees given for that set of paths. If *classes of service* (CoS) are considered within each aggregate, then we need to do something in order to individually meet the QoS guarantees for each CoS on each path. As discussed in chapter 3, we have two approaches to solve this issue. The first approach will still consider a commodity as all the traffic to be transported from node m to node n. In this case, the problem stays the same as before, where we need to establish the optimal layout for a given set of demands, and the QoS guarantees are globally met for all classes of services on each path through the constraints introduced in the MPSFAP problems. Individual relative guarantees could be given to each individual CoS within each path through service differentiation mechanisms (e.g. using Diffserv-aware LSPs). The second approach considers instead all the traffic aggregate for a single class of service to be transported between node m and node n as a commodity. Here, each path in the set of paths connecting a given node pair will transport a fraction of the traffic of a single CoS demanded between that node pair, so that individual QoS guarantees can be given to each CoS for each node pair.

The first approach is trivial from the point of view of problem formulation. The second approach needs a little bit of work to extend the model to multiple classes of service. In this section, we present the model extensions needed to formulate MPSFAP for multiple classes of service. It is worth noting that the extended problem will generate even larger instances than those generated for a single class in a particular network topology. This makes the development of heuristics even more interesting in the context of dimensioning IP next generation networks, where service differentiation and individual QoS guarantees are a strong requirement.

Formally, let us suppose that each pair q has now a demand composed of one or more service classes. Each pair can now originate and terminate more than one *commodity*. Let us extend the demand matrix to include a column for each class l indexed $1, \ldots, L$. The demand is now represented by a matrix $\mathbf{D}$:

$$\mathbf{D} = \begin{pmatrix} d_{1,1} & \ldots & d_{1,L} \\ d_{2,1} & \ldots & d_{2,L} \\ \vdots & \vdots & \vdots \\ d_{Q,1} & \ldots & d_{Q,L} \end{pmatrix} \tag{4.27}$$

The path flow matrix **B** is now modified to include the flow associated to each class of service:

$$\mathbf{B} = \left(\begin{array}{ccc} b_{1,1}^{1} & \dots & b_{1,L}^{1} \\ \vdots & \vdots & \vdots \\ b_{1,1}^{K_1} & \dots & b_{1,L}^{K_1} \\ \hline \vdots & \vdots & \vdots \\ \hline b_{Q,1}^{1} & \dots & b_{Q,L}^{1} \\ \vdots & \vdots & \vdots \\ b_{Q,1}^{K_Q} & \dots & b_{Q,L}^{K_Q} \end{array} \right) \tag{4.28}$$

The weight matrix **W**, is here a $K \times L$ matrix containing the weights associated to the paths connecting a pair q for a given commodity $d_{q,l}$. Calculating the weights for MPSFAP using (4.9) will produce equal path weights for all classes of service demanded between a given pair. Economic factors and particular service needs can be introduced through the choice of path weights (e.g. shortest paths in terms of number of hops would be preferred for classes associated with real time service). Accordingly, the matrix **H** is extended to indicate whether a given path a_q^k is used (entry $h_{q,l}^k = 1$) or unused (entry $h_{q,l}^k = 0$) by the commodity $d_{q,l}$. Similar extensions are made to the matrixes $\mathbf{\Theta}$ and $\mathbf{\Delta}$. **X** must now be calculated as $\mathbf{A} \cdot \mathbf{B} \cdot \mathbb{1}_{\mathbf{L}}$, where $\mathbb{1}_{\mathbf{L}}$ is a L-dimensional column matrix with all its entries set to 1. MPSFAP is extended to a Minimum Path Set and Flow Allocation Problem with Multiple Classes of Service (MPSFAPQ).

A MPSFAPQ1 problem can be then defined, where $\mathbf{W}_{(\mathbf{l})}$ and $\mathbf{H}_{(\mathbf{l})}$ are the l column of the corresponding matrixes **W** and **H**. Similarly, $\delta_{q,l}$ is a limit in the quantity of flow acceptable on one path for a given commodity. As before, constraints (4.33) impose a limit on any path to the quantity of flow for a particular CoS and node pair. When dimensioning for multiple classes of service, traffic engineering can benefit of imposing such constraints in order to achieve the planified bandwidth limits for each class on critical links. The formulation in MPSFAPQ is compatible with the Diffserv aware traffic engineering in MPLS [11].

A MPSFAPQ2 problem can be similarly defined to take care of the delay variation among paths transporting the same commodity. The objective function (4.8) can be then extended to include multiple classes of service:

$$\text{Minimize} \left[\alpha \sum_{q=1}^{Q} \sum_{k=1}^{K_Q} \sum_{l=1}^{L} w_{q,l}^k h_{q,l}^k + \beta \sum_{q=1}^{Q} \sum_{k=1}^{K_q} \sum_{l=1}^{L} h_{q,l}^k \sum_{i=1}^{M} \frac{\lambda a_{q,i}^k}{C_i - x_i} \right] \tag{4.36}$$

MPSFAPQ 1
Reduced Complexity Layout with End-to-End Delay Guarantees - QoS Extension: Multiple Classes of Service

Given:

$$\mathbf{A}, \mathbf{C}, \mathbf{D}, \mathbf{\Theta}, \mathbf{W}$$

minimize :

$$\sum_{l=1}^{L} {}^{t}\mathbf{W}_{(\mathbf{l})} \cdot \mathbf{H}_{(\mathbf{l})} \tag{4.29}$$

subject to:

$$\mathbf{A} \cdot \mathbf{B} \cdot \mathbb{1}_{\mathbf{L}} \leq \mathbf{C} \tag{4.30}$$

$$\mathbf{R} \cdot \mathbf{B} = \mathbf{D} \tag{4.31}$$

$$h_{q,l}^{k} \sum_{i=1}^{M} \frac{\lambda a_{q,i}^{k}}{C_i - x_i} \leq \theta_{q,l} \qquad l = 1, \ldots, L; k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.32}$$

$$0 \leq b_{q,l}^{k} \leq h_{q,l}^{k} \delta_{q,l} \qquad l = 1, \ldots, L; k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.33}$$

$$h_{q,l}^{k} \in \{0, 1\} \qquad l = 1, \ldots, L; k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.34}$$

$$b_{q,l}^{k} \geq 0 \qquad l = 1, \ldots, L; k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{4.35}$$

4.8. Conclusions

The problem of obtaining optimal routing schemes is a well known issue in the field of network flow problems with application to communication and transport networks. In those contexts, the usual objectives are related to the use of paths in order to globally ensure a minimum congestion when transiting the network. When dealing with telecommunication networks such objectives are usually kept. Given that in a pure IP environment it is practically impossible to find a unique metric allowing to implement a fully optimal general routing scheme using a routing protocol [40], we resort to source routing schemes. In particular, layouts established as MPLS LSPs allow the implementation of any general optimal routing. Paths are calculated and established by the management system. However, while a minimum congestion layout is a common optimal routing objective, it is not representative of the cost of operations and maintenance incurred by an operator when establishing a given routing scheme through a source routed layout. With this in mind, from a set of layouts capable of transporting a given demand within certain QoS bounds given by the underwritten service contracts, an operator would chose those which guarantee a minimum cost for their operational structure.

A first contribution in our work is the definition of a *minimum complexity* layout as the optimization objective. Whenever the QoS guarantees are met, considering the operator's objectives, the optimal layout is the one which minimizes the quantity of *elements* to manage during operation: the quantity of links and paths that the layout globally uses to transport a given demand. Important work on the subject consider the optimization of congestion based objective functions without considering end-to-end QoS guarantees (such as path-delay), both in the on-line and off-line contexts [15, 22, 23, 45, 46, 47]. The main interest being the development of algorithms and heuristics to solve the resulting complex problem. End-to-end path delay constraints have been considered so far in the context of multiconstrained route selection [48], routing from one source to all destinations [49, 50], or optimizing virtual circuit layouts [24], but the cost functions still consider congestion based objectives, and the path-delay is generally modelled as a hop-count constraint. The issue of layout complexity for MPLS layouts has particularly been addressed in [88], where restrictions to the number of paths allowed to traverse any given link are imposed as a way of reducing the quantity of labels needed at every hop. The resulting problem can be formulated as a MIP (Mixed Integer Linear Program), as the end-to-end path delay constraints are not included. This allows to exactly solve large instances of the problem by techniques such as Lagrangean Decomposition. The definition of end-to-end path delay constraints as bundling constraints in the problem formulation constitutes the second contribution in our work, as the end-to-end path delay associated to hop-count is not realistic in the context of MPLS networks as it is in the context of optical network design due to its coarse bandwidth granularity and broadband nature. However, some simplifications in the path delay constraints are necessary to implement suitable heuristics, as we describe further in Chapter 5. As discussed, the non-linearities in our problem formulation are introduced by the end-to-end path delay constraints, which constitute one of the main contributions in our work. We could try to linearize the non-linear constraints in order to be able to apply Lagrangean Decomposition. To do so, we would need to linearize a large number of delay constraints, as a different piecewise linear function is needed for each different capacity. Although this is possible to do, we prioritize then the choice of offering a fine granularity to the layout design, and then we need to develop the heuristic methods presented in next chapter to approximately solve the problem for large networks.

5. Contribution to the Development of Heuristics for Solving the Minimum Path Set and Flow Allocation Problems (MPSFAP)

5.1. Exact Methods for Solving MINLP Problems

The *Minimum Path Set and Flow Allocation Problems* (MPSFAP) are formulated in Chapter 3 as Mixed-Integer Non-Linear Programming (MINLP) problems. MINLP problems are $\mathcal{NP}$-*complete*, what makes its resolution intractable for large instances. For computationally tractable problem sizes, *exact* or *deterministic* algorithms can be used to find optimal solutions. When the problem size becomes larger, we must resort to *heuristic* methods, which yield approximated optimal solutions. Deterministic algorithms use implicit enumeration and tree search rules, with evaluation techniques to determine whether a given solution is optimal or not. Deterministic algorithms commonly used for exact solution of MINLP problems range within one of the following types:

Branch and Bound (B&B)

The B&B algorithm for MINLP problems [51, 52] is based on the same ideas as B&B for solving MILP problems. The first step is to solve the problem generated by relaxing the integrality condition on the variables. If the solution of that problem fulfils all integrality conditions the whole problem is solved. Otherwise, in a minimization problem the relaxed problem provides a lower bound (of course only if the global minimum can be determined) and the search tree is built up. A feasible integer solution provides an upper bound. A major drawback of B&B applied to MINLP problems is that nodes deeper in the tree cannot benefit so greatly from information available at previous nodes as is the case in MILP B&B using the dual simplex algorithm. A variant of B&B method called Extended Cutting Plane (ECP) [57] does not solve the NLP subproblems. Instead, it relies exclusively on successive linearizations.

Generalized Benders Decomposition (GBD)

Generalized Benders Decomposition [53] divides the variables into two sets: complicating and non-complicating variables. In MINLP models the class of complicating variables is made up by the discrete (usually binary) variables. Then the algorithm generates a sequence of NLP sub-problems (produced by fixing the binary variables) and solves the

so-called MILP Master problems in the space of the complicating variables. The NLP sub-problems yield upper bounds for the original problem while the MILP Master problems yield additional combination of binary variables for subsequent NLP sub-problems. Under convexity assumptions the Master problems generate a sequence of lower bounds increasing monotonically. The algorithm terminates if lower and upper bounds equal or cross each other.

Outer Approximation (OA)

Outer Approximation [54, 55] also consists of a sequence of NLP sub-problems produced by fixing the binary variables generated by MILP Master problems. The significant difference is how the Master problems are defined. Algorithms based on OA describe the feasible region as the intersection of an infinite collection of sets with a simpler structure (e.g. polyhedra). In OA the Master problems are generated by *outer approximations* (e.g. linearizations or Taylor series expansions) of the nonlinear constraints in those points which are the optimal solutions of the NLP subproblems (i.e. a finite collection of sets). The key idea is to solve the MINLP with a much smaller set of points (i.e. tangential planes). In convex MINLP problems, a superset of the feasible region is established. Thus, the OA Master problems (MILP problem in both discrete and continuous variables) produce a sequence of increasing lower bounds. The termination criterion is the same as in GBD.

The problem subclasses of MINLP, *Mixed Integer Linear Programming* (MILP) and *Non-Linear Programming* (NLP), present different degrees of difficulty. MILP are combinatorial problems for which branch and bound techniques with linear programming (LP) relaxation is often proven to be sufficient. NLP problems can only be solved iteratively. NLP problems, however are usually easier to solve than MILP problems. MINLP combines all the difficulties of its subclasses, adding other difficulties of its own: while for convex NLP problems the local minimum is identical to the global minimum, the same property does not hold for convex MINLP problems; even when assuming convexity in the MINLP problem could be useful, it constitutes no guarantee of finding a global minimum.

In recent years a new trend has emerged in the formulation and solution of mixed-integer optimization problems: the Generalized Disjunctive Programming (GDP). The basic idea behind GDP is to use boolean and continuous variables, and to formulate the problem with an objective function and subject to three type of constraints: global inequalities independent of integer decisions, disjunctions that are conditional constraints involving an OR operator, and pure logic constraints that involve only the boolean variables. The interest in this method resides in that it simplifies the problem formulation. Further details on GDP are out of the scope of this thesis, as our focus is on models for network optimizations in

the context of dimensioning and reconfiguration, as well on heuristics allowing to obtain good enough approximated solutions in times compatible with the dynamic control of the network. However, it constitutes an interesting research direction for extending the present work.

We have presented here some details about the deterministic methods and some well-known algorithms to solve general MINLP problems. The MINLP solver used in Chapter 4 to obtain results for the small network topologies NET1 and NET2 implement some of the above described techniques [56]. The different methods used by the deterministic solvers lead in general to the need of solving a large number of NLP subproblems when applied to the MPSFAP formulated here, since the number of integer variables increases with the number of paths, which grows exponentially with the size and density of the network. Linearizing the constraints would help in reducing the NLP subproblems to LP subproblems (enabling more efficient implementations of B&B for instance), but the number of subproblems to solve would still be prohibitively large considering our objective. Our main interest lies then in the development of algorithms implementing heuristics capable of avoiding the need to solve such a large number of subproblems. It is worth noting that our main objective however, is not to develop algorithms to improve existing methods and algorithms largely studied and implemented in the field of operations research, but to find heuristics capable of yielding good approximations to the optimal solution, which will allow us to study the proposed problems. Two heuristic methods to intelligently explore the solution space without solving a large number of subproblems are presented next. The first uses Tabu Search (TS) techniques to explore the solution space, while the second modifies the well-know flow deviation algorithm used in the classical literature [64, 31] and a particular implementation [65] to solve our MINLP formulation of MPSFAP problems. In Chapter 8, a further improvement to the *Modified Flow Deviation* (MFD) algorithm produces solutions for large networks for the dimensioning and reconfiguration problem defined in Chapter 7.

5.2. Meta-Heuristics: Tabu Search Methods

Optimization algorithms based on *meta-heuristics* have recently gained interest because they are able to cope with problem instances of large size, overcoming the complexity issues of most problems found in practical applications. Among others, Simulated Annealing (SA) [58], Genetic Algorithms (GA) [59] and Tabu Search (TS) [60, 61] have been proposed to tackle multicommodity flow problems. The computed solution however is not guaranteed to be optimal. That is why it is necessary to check the accuracy of the solutions provided by such a heuristic based algorithm and to compare them with exact solutions on small size problems. A detailed description of the TS algorithm can be found in [60].

The TS basic idea consists in exploring the complex space of solutions until a defined number of iterations is reached or until a specific cost criterion is satisfied. The exploration starts with an initial solution computed by another algorithm (e.g. an initial solution randomly generated). At each iteration, TS computes a set or *neighborhood* of solutions derived from the current solution via *perturbations* applied to this solution. All the solutions of the neighborhood are evaluated (i.e. assigned a value according to a cost function) and the best one is selected as the new current solution. In order to prevent the algorithm from cycling along the same series of current solutions, a *tabu list* is maintained, which contains a number of last visited solutions that cannot be chosen as long as they belong to this list. This allows the algorithm to choose a solution worse than the current one, allowing it to escape from the local minima possibly found during the search.

Formally, let $\mathcal{N}(s)$ be the neighborhood of a given solution s in the space of solutions $\mathcal{S}$. $\mathcal{N}(s)$ is obtained by applying an elementary transformation to s. The method can be viewed as an iterative technique which explores a set of problem solutions by repeatedly moving from one solution s to another solution s' located in the neighborhood $\mathcal{N}(s)$ of s. The moves aim at efficiently reaching a solution which qualifies as *good* (i.e. optimal or near-optimal) with respect to some objective function $f(s)$ to be minimized. Using classic descendant methods to search the solution space (i.e. choosing a solution s' for which $f(s') < f(s)$) will lead to get trapped in local minima for non-convex objectives; the local minima will depend upon the way the initial point was generated. To avoid getting trapped in local minima, the search procedure must allow some solution for which $f(s') > f(s)$, so as to explore different regions of the solution space. But accepting a solution s' that may be worse than s may produce cycling. To avoid this, TS adds a *memory* of the visited solutions, forbidding the search process to visit a solution which is in a *tabu list* of already evaluated solutions. The concept of memory can be formally stated by saying that the neighborhood also depends on the iteration, so we denote it now by $\mathcal{N}(s, j)$, being j the iteration in which s was visited. We can now write a *pseudo* code with the general TS idea:

```
Generate an initial solution s ∈ S
s* := s
j := 0
while the stopping condition is not met do
   j := j + 1
   Generate V* ⊆ N(s, j)
   Choose the best s' in V*
   if f(s') < f(s*) then
      s* := s'
   end if
end while
```

Algorithm 5.1: Pseudo-code for the General Tabu Search Method

Practically, the concept of memory will be implemented through a *tabu list* T of already visited solutions. There may be several possible stopping conditions, but the simplest would be some logical combination of the following:

- An optimal solutions is found.
- $\mathcal{N}(s, j+1) = \varnothing$.
- k is greater than the maximum number of iterations allowed.
- The number of iterations performed since last s^* changed is greater than a specified maximum.

5.3. A Tabu Search Heuristic Approach Applied to the MPSFAP Problems

We restrict to the case of a single class of service to define the TS heuristics applied to the MPSFAP problems [**?**]. The heuristics can easily be extended to multiple classes of service as defined in MPSFAPQ problems in Chapter 4. Three problem-specific elements must be defined to implement a particular TS heuristic for the MPSFAP problems[1] [62]:

5.3.1. Initial Solution

An initial solution must be computed. For the MPSFAP problems, the initial layout is the one obtained when taking the shortest path in terms of number of hops (although other metrics could be used) for every pair of nodes. For each shortest path $\hat{a}_q$ connecting node q, the whole demand d_q is assigned as path flow. The resulting layout can be *feasible* or *infeasible*, and it will be evaluated according to the objective function described below[2]. A solution s is defined in this context as a set of Q tuples or coordinates with K_q components:

$$s = \left\{ \left(b_q^1, \ldots, b_q^{K_q} \right) : q = 1, \ldots, Q \right\} \tag{5.1}$$

5.3.2. Perturbation mechanism

A perturbation mechanism is necessary to generate a neighborhood $\mathcal{N}(s)$ of the current solution s. A neighbor solution $s_1 \in \mathcal{N}(s)$ is calculated by randomly choosing a source-destination pair q, and then rearranging some of the path flow values b_q^k. The way path flows are changed for the node pair q is also random. The new flow distribution meets the

[1] The TS heuristics were developed in the context of the internship of V.Friderikos at the INFRES department, Telecom Paris (ENST) (financed by the VTHD project [63]), co-directed by S.Beker and N.Puech. Work based on a previous implementation due to T.Vergnaud

[2] The problem of obtaining a feasible layout is also $\mathcal{NP}$-*complete*, and it can be viewed as the constraint satisfaction subproblem in MPSFAP

demand constraints but not necessarily meets the link flow and path delay constraints. The number of node pairs which are to have their path flows changed when generating each neighbor is a parameter of the algorithm. The number of neighbors to generate from the current solution is also a parameter of the algorithm. The first parameter, the number of node pairs to have their path flows changed is a compromise between the aggressiveness of the solution space exploration and the computation complexity of the algorithm. Hence, the neighbor generation process leads to new candidate solutions that may be considered as *valid* (i.e. solutions that meet all the constraints) or *invalid* (i.e. solutions that do not meet the link flow or path delay requirements).

5.3.3. Evaluation Functions

A cost function $\mathbf{f} : \mathcal{S} \rightarrow \mathbb{R}$ mapping elements in the solution space $\mathcal{S}$ to real numbers must be defined so as to allow comparison of the different solutions in the neighborhood $\mathcal{N}(s)$ of the current solution s. Since we allow infeasible layouts as initial solution, we must ensure that the evaluation of invalid solutions in the neighborhood of the current one will drive the exploration of the solution space towards feasible or valid solutions. We use a cost function that evaluates distinctly valid and invalid solutions. Indeed, not all invalid solutions are the same, and we must evaluate them differently. According to the constraint being violated in the solution, we can distinguish two types of invalidity:

Type-1 Invalidity: A solution s is invalid because at least one of the *link capacity* constraints (4.13)/(4.20) is violated.

Type-2 Invalidity: A solution s is invalid because all the capacity constraints are met, but at least one of the *path delay* constraints (4.15)/(4.22) is violated.

Practically, we want the evaluation to be as computationally inexpensive as possible. By evaluating only capacity constraints first, we only perform M invalidity checks (there are M links in the network), if we found the solution to be type-1 invalid, then no further evaluation must be performed. If we found the solution not to be type 1-invalid, then further evaluation is necessary to decide wether it is type-2 invalid or valid. K type-2 invalidity checks are performed, where K is the total number of paths in the arc-incidence matrix A (K can be a very large number). Valid solutions are evaluated using the corresponding objective functions for MPSFAP1 and MPSAP2 (depending on the problem to be solved). A given solution s is evaluated as follows:

$$f(s) = \begin{cases} \text{MPSFAP1}(s)/\text{MPSFAP2}(s) & \text{if } s \text{ is valid} \\ g_1(s) & \text{if } s \text{ is } \textit{Type}\text{-1 } \textit{Invalid} \\ g_2(s) & \text{if } s \text{ is } \textit{Type}\text{-2 } \textit{Invalid} \end{cases} \tag{5.2}$$

we rewrite the objective functions MPSFAP1 and MPSFAP2 here for convenience:

$$\text{MPSFAP1}(s) = \sum_{q=1}^{Q} \sum_{k=1}^{K_q} w_q^k h_q^k \tag{5.3}$$

$$\begin{aligned} \text{MPSFAP2}(s) &= \alpha \sum_{q=1}^{Q} \sum_{k=1}^{K_q} w_q^k h_q^k + \beta \sum_{q=1}^{Q} \sum_{k=1}^{K_q} h_q^k \sum_{i=1}^{M} a_{q,i}^k \frac{\lambda}{C_i - x_i} \\ &= \sum_{q=1}^{Q} \sum_{k=1}^{K_q} h_q^k \left(\alpha w_q^k + \beta \sum_{i=1}^{M} a_{q,i} \frac{\lambda}{C_i - x_i} \right) \end{aligned} \tag{5.4}$$

with all variables defined as before (Chapter 4). We are interested in ensuring that evaluation for type-1 invalid solutions will be always greater than evaluation for type-2 invalid solutions. At the same time, we must ensure that evaluation for type-1 or type-2 invalid solutions is always greater than evaluation for valid solutions using either MPSFAP1 or MPSFAP2 evaluation. We know that MPSFAP2 evaluations are always greater than evaluations of the same solution with MPSFAP1 ($\alpha > 0$ and $\beta > 0$ by definition). We can then establish the following relationship:

$$g_1(s) > g_2(s) > \text{MPSFAP2}(s) > \text{MPSFAP1}(s) \tag{5.5}$$

In this way, valid solutions will be preferred over type-2 invalid solutions, and type-2 invalid solutions will be preferred over type-1 invalid solutions. At the same time, a mechanisms to provide a slope within type-1 invalid solutions must be provided, so as to guide the exploration of the solution space towards type-2 invalid or valid solutions. The evaluation of type-1 invalid solutions can then be defined as:

$$\begin{aligned} g_1(s) &= \sum_{i=1}^{M} g_1'(x_i) + \sum_{q=1}^{Q} \sum_{k=1}^{K_q} w_q^k + \sum_{q=1}^{Q} \sum_{k=1}^{K_q} 1 \\ &= \sum_{i=1}^{M} g_1'(x_i) + \sum_{q=1}^{Q} \sum_{k=1}^{K_q} \left(w_q^k + 1 \right) \end{aligned} \tag{5.6}$$

where $g_1'(x_i)$ is defined as:

$$g_1'(x_i) = \begin{cases} \frac{x_i - C_i}{C_i} & \text{if } x_i > C_i \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

for every link $i = 1, \ldots, M$ in the network. The first term in (5.6) provides a *degree of invalidity* for the type-1 invalid solutions. It measures the degree of violation in the link capacities constraint in order to provide a slope towards type-2 invalid or valid solutions. Similarly, The evaluation of type-2 invalid solutions is defined as:

$$\begin{aligned} g_2(s) &= \sum_{q=1}^{Q} \sum_{k=1}^{K_q} h_q^k g_2'(\tau_q^k) + \sum_{q=1}^{Q} \sum_{k=1}^{K_q} w_q^k \\ &= \sum_{q=1}^{Q} \sum_{k=1}^{K_q} \left(h_q^k g_2'(\tau_q^k) + w_q^k \right) \end{aligned} \tag{5.8}$$

where $g_2'(\tau_q^k)$ is defined as:

$$g_2'(\tau_q^k) = \begin{cases} \frac{\tau_q^k - \theta_q}{\theta_q} & \text{if } \tau_q^k > \theta_q \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

for $k = 1, \ldots, K_q$ and $q = 1, \ldots, Q$ (all paths connecting a node pair and all node pairs). τ_q^k is the calculated delay for path a_q^k in the current solution. As with $g_1(s)$, the first term in (5.8) provides a *degree of invalidity* for the type-2 invalid solutions. It measures the degree of violation in the path delay constraints in order to provide a slope towards valid solutions. The second and third terms in (5.6) ensure that $g_1(s)$ is greater than an evaluation of any type-2 invalid solution, since the second term supposes that all paths are used, and third term supposes that all paths are overloaded in terms of $g_2(s)$ evaluation. Similarly, the second term in (5.8) ensures that an evaluation of any type-2 invalid solution will be greater than an evaluation of any valid solution, since it supposes that all paths are used.

Ensure: **generate_neighbor**(s)
Require: current solution $(s \in \mathcal{S})$
Require: number of node pairs to change $(q_{max} > 0)$
  $s' \leftarrow s$
  $q \leftarrow 0$
  $q_{changed} \leftarrow 0$
  $list_{changed} \leftarrow \varnothing$
  while $q_{changed} < q_{max}$ **do**
    $q \leftarrow random(0, \ldots, 1) \times Q$
    if $K_q \geq 2$ and $q \notin list_{changed}$ **then**
      change_path_flows(q, s')
      $q_{changed} \leftarrow q_{changed} + 1$
    end if
  end while
  Return s'

Algorithm 5.2: Tabu Search Heuristics Applied to MPSFAP: algorithm to generate a neighbor of the current solution

Ensure: **change_path_flow**(q, s')
Require: node pair to change $(q > 0 : q = 1, \ldots, Q)$
Require: candidate neighbor solution $(s' \in \mathcal{S})$
  $load\_left \leftarrow d_q$
  $k \leftarrow 1$
  while $k \leq K_q$ and $load\_left > 0$ **do**
    if $k = K_q$ **then**
      $b_q^k \leftarrow load\_left$
    else
      $b_q^k \leftarrow b_q^k + random(-0.5, \ldots, +0.5) \times (1 - \frac{w_q^k}{\max_k w_q^k}) \times load\_left$
      if $b_q^k < 0$ **then**
        $b_q^k \leftarrow 0$
      end if
      if $b_q^k > load\_left$ **then**
        $b_q^k \leftarrow load\_left$
      end if
    end if
    $load\_left \leftarrow d_q - b_q^k$
    $k \leftarrow k + 1$
  end while
  Return s'

Algorithm 5.3: Tabu Search Heuristics Applied to MPSFAP: algorithm to change path flows for a selected node pair

Algorithm 5.4 represents the *pseudo code* for our implementations of TS heuristics applied to the MPSFAP problems. The stopping rule is a combination of a maximum number of global iterations and a maximum number of *inefficient* iterations (i.e. the number of global

iterations since the the algorithm didn't produce an improvement in the evaluation). The parameter $neighbor^{max}$ defines the total number of neighbors to generate during the current iteration. The best neighbor of all generated neighbors is then evaluated and compared to the evaluation of the current solution. If it doesn't evaluate better than the current solution, it is put in the tabu list so it would not be visited again. If it evaluates better than the current solution, then it is adopted as the current solution. Algorithm 5.2 generates as many neighbors as defined by the parameter $neighbor^{max}$. For each neighbor, it selects as many node pairs to have their path flows changed as defined by the parameter q_{max}. A node pair can only have its path flows rearranged if it has more than one path in the set of paths connecting it. Indeed, it will not be possible rearrange the path flow for a unique paths, as it has to carry the whole demand d_q. The path flows for each of the selected node pairs are changed as described in the algorithm 5.3. Path flows are increased or decreased with equal probability. The quantity of flow to be increased or decreased on a given path is randomly calculated, with a relative weight according to the path weights of the paths in the set: paths with lower weights have higher probability of receiving or losing flow.

Require: maximum number of global iterations ($iteration^{max} > 0$)
Require: maximum number of global inefficient iterations ($iteration^{max}_{inef} > 0$)
Require: number of neighbors to generate ($neighbor^{max} > 0$)
  Generate an initial solution ($s \in \mathcal{S}$)
  $s^* \leftarrow s$
  $T \leftarrow \varnothing$
  $iteration \leftarrow 0$
  $iteration_{inef} \leftarrow 0$
  while $iteration < iteration^{max}$ and $iteration_{inef} < iteration^{max}_{inef}$ **do**
    for $neighbor = 1$ to $neighbor^{max}$ **do**
      $s' \leftarrow$ **generate_neighbor**(s)
      if $neighbor = 1$ **then**
        $\hat{s} \leftarrow s'$
      end if
      if $f(s') < f(\hat{s})$ and $s' \notin T$ **then**
        $\hat{s} \leftarrow s'$
      end if
    end for
    if $f(\hat{s}) \geq f(s)$ **then**
      $T \leftarrow \hat{s}$
      $iteration_{inef} \leftarrow iteration_{inef} + 1$
    else
      $s^* \leftarrow \hat{s}$
      $iteration_{inef} \leftarrow 0$
    end if
    $iteration \leftarrow iteration + 1$
    $s \leftarrow s^*$
  end while

Algorithm 5.4: Pseudo-code for the Tabu Search Heuristics Applied to the MPSFAP Problems

5.3.4. Evaluation of TS Heuristics for MPSFAP

To evaluate the TS heuristics for the MPSFAP problems, we run the algorithm for the set of 25 matrixes and network topologies NET1 and NET2 used in Section 4.6 in Chapter 4 . We then compare the results obtained from TS with the results obtained from the MINLP solver. In this context, since the MINLP solver yields exact optimal solutions, the total cost of the solutions constitute a reference for optimality. The number of global iterations of the TS algorithm is set to 20, while the number of inefficient iterations is set to 10. 200 neighbors are generated for the current solution in each iteration of the algorithm. We find that having the path flows of 2 node pairs changed when generating a neighbor from the current solution is a good tradeoff between the speed of progression in the solution space and computational effort needed.

Table 5.3 shows the average values of the objective function for the set of demand matrixes considered. $f(s)_{MINLP}$ corresponds to the value of the objective function obtained from the MINLP solver in Chapter 4, and $f(s)_{TS}$ the average evaluation obtained from the TS heuristics (for both problems MPSFAP1 and MPSFAP2). The evaluation function $f(s)$ become the objective function given by (4.6), with w_q^k calculated as in (4.9) when the problem is feasible. This allows us to compare the values of the objective functions obtained from the two methods directly for feasible problems. To establish a comparison between the results obtained from MINLP and TS, we define $f(s)\%_{TS}$ as follows:

$$f(s)\%_{TS} = \frac{f(s)_{TS} - f(s)_{MINLP}}{f(s)_{MINLP}} 100 \tag{5.10}$$

		MPSFAP1			**MPSFAP2**		
		$f(s)_{MINLP}$	$f(s)_{TS}$	$f(s)\%_{TS}$	$f(s)_{MINLP}$	$f(s)_{TS}$	$f(s)\%_{TS}$
NET1	$a = 3/p = 0.2$	16.70	18.68	11.86	16.73566	17.84925	6.91
	$a = 4/p = 0.2$	16.00	17.20	7.50	16.01753	16.30396	1.79
NET2	$a = 5/p = 0.2$	15.16	16.74	8.18	15.1943	16.07869	5.82
	$a = 6/p = 0.2$	14.60	15.88	8.77	14.62582	15.66635	7.11

Table 5.1: Quality of the Solutions from TS with respect to MINLP for MPSFAP1 and MPSFAP2

It can be seen from the values of $f(s)\%_{TS}$ that MPSFAP problems can be approximated by the TS heuristics with an acceptable error. Solutions obtained for MTDFAP in Chapter 4, evaluated with the objective functions of MPSFAP1 and MPSFAP2 yield close values in average than those obtained from TS. This means that, even when the approximations are good enough, we are obtaining layouts with TS that are not better in terms of the cost functions considered than those obtained when optimizing a classic performance objective such as total delay.

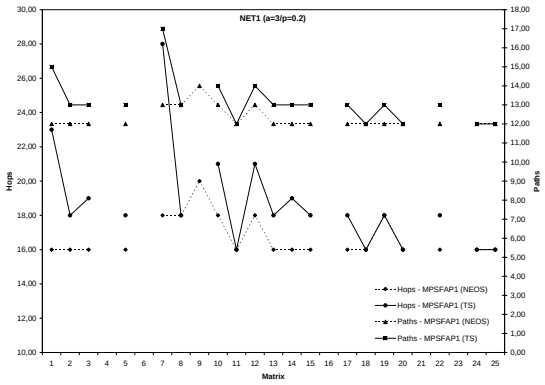
		Hops				
		MPSFAP1		**MPSFAP2**		**MTDFAP**
		MINLP	**TS**	**MINLP**	**TS**	**MINLP**
NET1	$a = 3/p = 0.2$	16.70	18.68	16.70	17.80	17.70
	$a = 4/p = 0.2$	16.00	17.20	16.00	16.28	16.40
NET2	$a = 5/p = 0.2$	15.16	16.74	15.16	16.05	16.32
	$a = 6/p = 0.2$	14.60	15.88	14.60	15.64	15.84

Table 5.2: Comparison of Average Quantity of Hops from MINLP and TS for MPSFAP1 and MPSFAP2

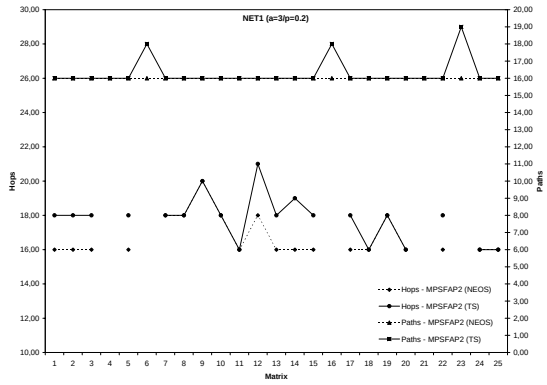
However, it can be seen that the approximation gets better as the size of the network increases. Indeed, as the network size increases (or the network gets more densely connected), the diversity of paths available also increases, offering a larger number of possible combi-

nations to be tried. In conclusion, as there is more room (solution space) to explore, the exploration becomes more effective and the solutions approach the optimal. Unfortunately, solutions for larger networks obtained from TS cannot be compared to those obtained with the reference solver MINLP, as problem size limitations arise with the last for networks larger than those already considered. However, solutions from TS for large networks can be compared with solutions from the heuristic algorithm presented next, which will be also referenced to the solutions obtained from the MINLP solver for NET1 and NET2 topologies.

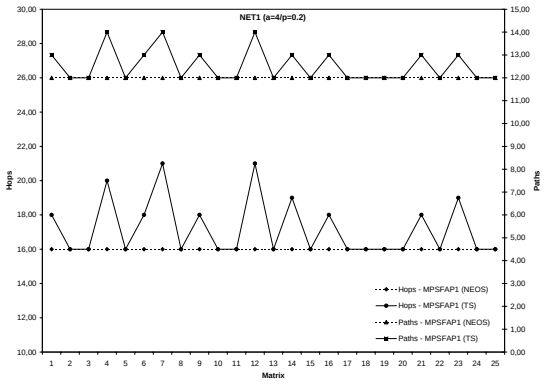
Tables 5.2 and 5.3 show the average hop-count and path-count obtained from TS and MINLP for all cases considered. Figures 5.1 and 5.2 show the compared results for each matrix. Points without value in the graphs represent infeasible matrixes.



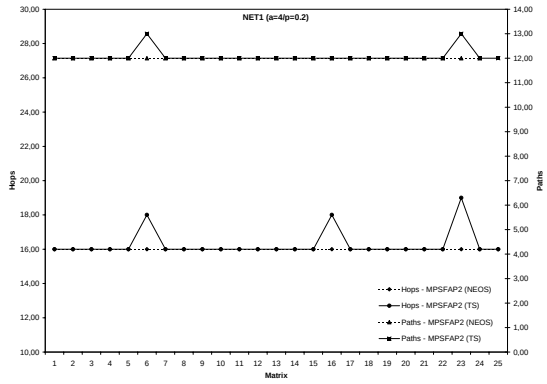
(a) MPSFAP1, $a = 3$



(b) MPSFAP2, $a = 3$



(c) MPSFAP1, $a = 4$



(d) MPSFAP2, $a = 4$

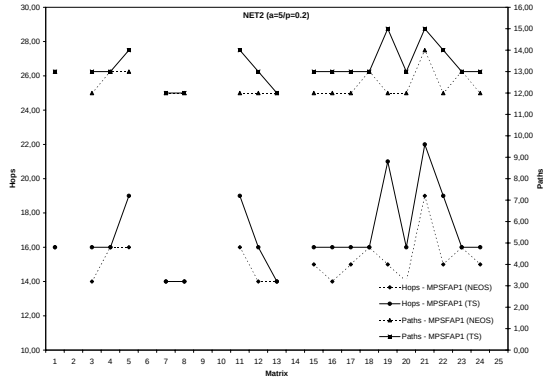
Figure 5.1: Results from MINLP Solver (NEOS) and Tabu Search (TS) for NET1

Finally, we evaluate the CPU time consumed in an Intel Pentium IV© running at 2.4 GHz and 512 Mb RAM. Computational times for NET1 topology average to 20 seconds, while for NET2 topology average to 25 seconds. Indeed, time increase doesn't directly relate

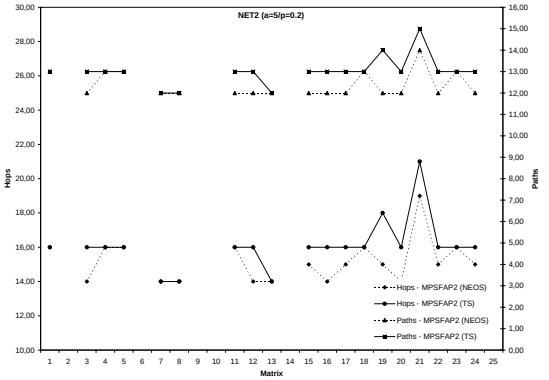
to combinatorial complexity at each iteration for TS as it does with the numeric solvers. At each iteration, TS algorithm must generate a number of neighbors, operation whose complexity is limited to the number of node pairs Q and the number of paths connecting the node pairs which are to be rearranged K_q.

		Paths				
		MPSFAP1		**MPSFAP2**		**MTDFAP**
		MINLP	**TS**	**MINLP**	**TS**	**MINLP**
NET1	$a = 3/p = 0.2$	12.30	13.16	12.35	12.85	12.85
	$a = 4/p = 0.2$	12.00	12.52	12.00	12.08	12.20
NET2	$a = 5/p = 0.2$	12.37	13.21	12.37	13.00	12.79
	$a = 6/p = 0.2$	12.12	12.84	12.12	12.72	12.52

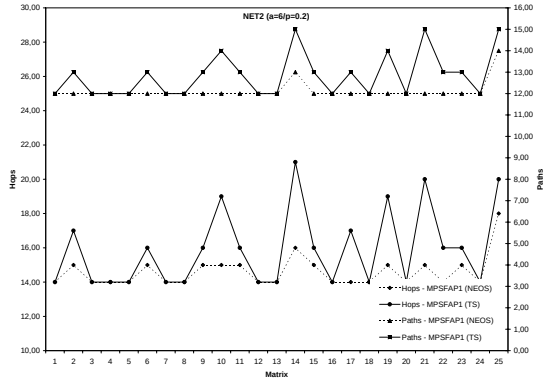
Table 5.3: Comparison of Average Quantity of Paths from MINLP and TS for MPSFAP1 and MPSFAP2



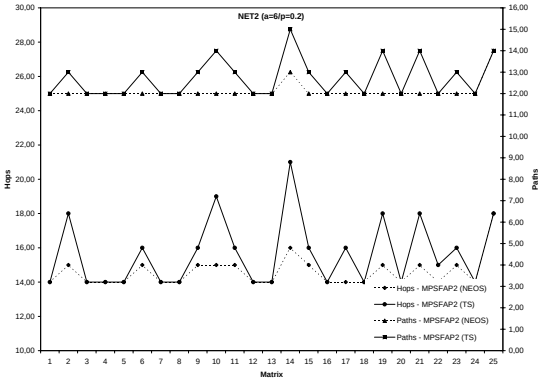
(a) MPSFAP1, $a = 5$



(b) MPSFAP2, $a = 5$



(c) MPSFAP1, $a = 6$



(d) MPSFAP2, $a = 6$

Figure 5.2: Results from MINLP Solver (NEOS) and Tabu Search (TS) for NET2

The complexity of the algorithm is then in the order of K the total number of paths in the network. This could also be a problem for large networks, as we will see in Chapter 6 when evaluating results for large topologies. Even when able to solve large problems (unlike solvers like MINLP using B&B techniques), TS will require increasingly big amounts of CPU time to cope with them. Our main objective is to find heuristics that would allow solving the dimensioning (and later the reconfiguration) problems in times compatible with network operation. In next section we develop such heuristics adapted to the nature of the MPSFAP1 problem.

5.4. Ad-Hoc Heuristics Based on The Flow Deviation Algorithm

Ad-Hoc heuristics take advantage of certain knowledge of the nature of the problem being solved. An ad-hoc heuristic is developed to solve the particular problem considered and results particularly efficient to solve it, contrary to meta-heuristics whose general procedure can be used to solve a broad range of problems with more or less efficiency. In the present section, we identify a set of particular characteristics related to the MPSFAP problems which help us in developing a particularly efficient algorithm based on the well known Flow Deviation Algorithm due to Frank and Wolfe [67]

5.4.1. The Flow Deviation Algorithm

Considering convex cost functions associated to each path in the network, a well known result says that the *optimal routing* results when flow travels along *Minimum First Derivative Length* (MFDL) paths for each node pair [64]. An obvious consequence is that a layout is *suboptimal* if some positive quantity of flow is travelling along a non MFDL path for some node pair. It follows that a way to improve a suboptimal layout would be to iteratively shift flow from non MFDL paths to MFDL paths. A method based on this idea would have to decide first on a *direction* of change and then on the *optimal* amount of flow to shift on each iteration. In general, a direction of change have to meet two requirements:

1. The direction of change has to be a *feasible direction* in the sense that if $\mathbf{B}$ is a feasible path flow vector, then for a positive ϕ, the path vector $\mathbf{B} + \phi\Delta\mathbf{B}$ obtained by shifting ϕ units of flow along the direction $\Delta\mathbf{B} = \left\{\Delta b_q^k\right\}$ is also a feasible path flow vector.

2. The direction of change has to be a *descent direction* in the sense that the cost function can be decreased by making small movements along the direction $\Delta\mathbf{B}$ from the current point $\mathbf{B}$.

The Frank-Wolfe (Flow Deviation) Method

The *Flow Deviation Method* [67] is a particular case of the Frank-Wolfe method to solve general, non-linear programming problems with convex constraints sets. It implements the philosophy of incremental changes along feasible descent directions to solve a minimum cost multicommodity flow allocation problem. Given a feasible path flow vector $\mathbf{B}$, we find a minimum MFDL path for each node pair. The first derivatives of the cost function $\mathbf{F}'(\mathbf{B})$ are evaluated at the current path flow vector $\mathbf{B}$ (we assume a convex cost function of the path flows)[3]. Let $\overline{\mathbf{B}} = \left\{\overline{b_q^k}\right\}$ be the vector of path flows that would result by routing all the demand d_q along the corresponding MFDL paths for each node pair. Let ϕ^* be the stepsize that minimizes $\mathbf{F}\left[\mathbf{b_q^k} + \phi(\overline{\mathbf{b_q^k}} - \mathbf{b_q^k})\right]$, $\phi \in [0,1]$. The new set of path flows is obtained by:

$$b_q^k \leftarrow b_q^k + \phi^*(\overline{b_q^k} - b_q^k) \tag{5.11}$$

for $k = 1, \ldots K_q$ and $q = 1, \ldots, Q$. The process continues until some stopping rule is met (e.g. a number of iterations without improvement in the cost function). The *flow deviation method* is shown to reduce the value of the cost function to its minimum in the limit [64], though the convergence rate near the optimum tends to be very slow. Mathematically, the feasible descendent direction must meet the following requirements:

$$\sum_{k=1}^{K_q} \Delta b_q^k = 0 \qquad \text{for} \quad q = 1, \ldots, Q \tag{5.12}$$

and

$$\sum_{q=1}^{Q} \sum_{k=1}^{K_q} \frac{\partial \mathbf{F}(\mathbf{B})}{\partial b_q^k} \Delta b_q^k < 0 \tag{5.13}$$

Requirement (5.12) is a flow conservation condition for the set of paths connecting a given node pair: the total flow shift for the path set must be cancelled. Requirement (5.13) indicates that the flow shift must be done in the sense of decreasing the cost function. In other words, for all non shortest paths (in the sense that $\frac{\partial \mathbf{F}(\mathbf{B})}{\partial b_q^k} > \frac{\partial \mathbf{F}(\mathbf{B})}{\partial \overline{b_q^k}}$) connecting the node pair q, we require that $\Delta b_q^k \leq 0$; and for at least one shortest path of the same node pair, we require that $\Delta b_q^k > 0$. This conditions are very important for the design of any algorithm based on the flow deviation method.

[3] The Modified Flow Deviation Algorithm proposed in Section 5.5 works out a way of adapting this method to the cost functions proposed for the MPSFAP problems, which are not dependent on the path flows b_q^k

5.5. A Modified Flow Deviation (MFD) Algorithm for Solving the MPSFAP Problems

The characteristic property of the flow deviation method is that flow is shifted to the shortest paths in *equal* proportions (the optimal proportion ϕ^*) for each node pair. In the context of multicommodity flow problems, flow is shifted from non-shortest paths to shortest paths in equal proportion for *all* node pairs [65]. A different approach, called *gradient projection methods* [64], shift flow from non-shortest to shortest paths in unequal proportions, making convergence to the optimum quicker, but still shifting flow for all node pairs. Besides, the direction of change are chosen following feasible descent directions. These characteristics arise many issues when trying to adapt the flow deviation method to our MPSFAP problems:

1. If a set of currently used paths in each iteration **H** is maintained, according to the particular implementation in [65], a path is added to **H** whenever it is determined to be the shortest (in the MFDL sense) for a given node pair and it is not already in **H**. Conversely, a non-shortest path is leaving **H** whenever its corresponding path flow $b_q^k = 0$ after a flow shift to a shortest path. The rate at which paths are incorporated to **H** is likely to higher than the rate at which paths are leaving it, due to:

   - The cost of each path is generally a convex function of link usage. As a consequence, the currently unused paths are likely to present lower costs than currently used paths when the algorithm is not iterating near the optimum.
   - A shortest path is being calculated for each node pair in the current iteration.
   - The flow shifts are made in equal proportion for all paths. This proportion is calculated in a way that it is the minimum ϕ^* to make at least one non-shortest path to have its flow lowered to 0. The number of paths leaving **H** in each iteration is unlikely to be bigger than 1.

2. The feasible descent directions assume that the starting point is an *already feasible* path flow set, and that convex differentiable functions are being used.

   - We have stated in Section 4.6 in Chapter 4 that it is also an $\mathcal{NP}$-*complete* problem to find a feasible path flow for MPSFAP problems. We have interest in being able to start from any random solution and to make the algorithm converge towards feasible optimal solutions.
   - The objective functions being proposed in MPSFAP problems are not convex in the link flow, as we are not interested in penalizing link usage as in the classic literature. Our objective is to minimize the total number of hops (indirectly the number of paths) used to set up a layout, while meeting the QoS constraints established in Chapter 4.

A major contribution in this section consists in the introduction of the appropriate modifications to the flow deviation algorithm in order to address the above mentioned issues, and to adapt it to our particular MPSFAP problems. The key modifications introduced to the basic flow deviation algorithm as implemented in [65] can be summarized as follows:

1. We shift flow from non-shortest to shortest paths for only one pair of nodes in each iteration. This allows us to limit the number of paths added to $\mathbf{H}$ to almost 1 per iteration. This also allows the algorithm to increase the proportion ϕ^* of flow that is to be shifted, and the probability that at least one of the non-shortest paths in the path set will have its path flow b_q^k lowered to 0, leaving $\mathbf{H}$.

2. In order to be able to start from non-feasible layouts, we will decide the direction of change following only descent directions in the sense of decreasing the cost function, but allowing a non-feasible path set $\mathbf{B}$ in the current iteration.

3. The cost function has to be adapted in consequence as we did for the Tabu Search heuristics, so as to present higher evaluations for overloaded paths than for non-overloaded paths, as well as to present a slope towards feasible path flows. Path length is then determined as a function of the overload proportion to the link capacity for the overloaded component links.

4. We simplify the capacity and path delay constraints, combining them into a unique capacity-delay constraint per link.

In what follows, we present the Modified Flow Deviation Algorithm (MFD), including the detailed description of the issues mentioned above. We restrict to the case of a single class of service to define the MFD algorithm applied to the MPSFAP problems. The heuristics can easily be extended to multiple classes of service as defined in MPSFAPQ problems in Chapter 4. In particular, we address the MPSFAP1 problem, as the method takes advantage of the nature of that particular problem.

Algorithm 5.5 shows the general procedure for the MFD method. The general procedure basically starts assigning all the demands to the set of shortest paths (in terms of hop-count). If constraints are met, then the algorithm stops and the optimal solution is found. If the initial set is infeasible, then path weights are recalculated as indicated in the corresponding procedure (in terms of hop-count and link overload) and a new set of shortest paths is calculated using those new calculated path weights. A *shift direction* $\mathbf{\Delta B}$ is then calculated. The node pair to be shifted is decided basically searching for the one which would produce the higher benefit (maximum cost function reduction) if its paths are having their flows shifted by the maximum allowed proportion ϕ_q for that node pair. The maximum proportion of change is calculated on a node pair basis, and it results from the minimum flow that can

be shifted out from the least loaded or shifted in to the critically loaded component link for all paths connecting q. After the flow shift, the set of currently used paths $\mathbf{H}$ is updated. If $\mathbf{\Delta B} = 0$, then no shift direction could be found and the algorithm finishes. The algorithm ensures that $\mathbf{\Delta B} = 0$ either if the current solution is valid or if there is no more candidate node pairs.

For a valid solution, the procedure is likely to have found the best approximation to the optimal, since the flow shift proportions are calculated in such a way that paths are being *exchanged* before sharing the load. A less loaded path is entering $\mathbf{H}$ while a more loaded one is leaving it, keeping the quantity of used paths to a minimum (the choice of the node pair to shift is calculated in a way that path swaps are preferred to load sharing during the algorithm iterations). Since the algorithm decides flow shift proportions looking at the component links loads, we need a way of translating path constraints to link constraints.

5.5.1. Path Delay Constraints Simplification

We assume that the end-to-end path delay constraint can be transformed into a link delay constraint for all the component links of the path. This transformation allows us to combine the new link delay and the previous capacity constraints into a unique constraint. In order to operate the transformation of the end-to-end path delay constraint into a link delay constraint, we take into account the path flows traversing each link. We set the link delay constraint to the most restrictive path delay constraint on the paths traversing it, individualized to that link.

In other words, let θ_q be the delay constraint imposed to every path transporting the commodity q. Let w_q^k be the weight associated to the path a_q^k, calculated as its hop-count. Making the simplification that each component link of a path equally contributes to the total path delay, we can write the delay constraint imposed on a any link i by a particular path a_q^k traversing it as $\frac{\theta_q}{w_q^k}$. Considering all paths traversing that particular link, the delay constraint imposed on it can be calculated as:

$$\theta_i = \min_q \min_k \left\{ a_{q,i}^k \frac{\theta_q}{w_q^k} \right\} \tag{5.14}$$

Assuming that conditions in Section 4.3 in Chapter 4 hold, the link delay constraint will reduce the bandwidth usage on a link with capacity C_i by the quantity $\frac{\lambda}{\theta_i}$, where λ is the average packet size. Figure 5.3 shows an example of path delay and capacity to link delay constraint simplification.

Require: $\mathbf{A}, \mathbf{\Theta}, \mathbf{W}, \mathbf{D}, \mathbf{C}$ as defined in Chapter 4
Ensure: $\mathbf{B}, \mathbf{H}$ or INFEASIBLE
  $\mathbf{B} \leftarrow 0$
  $\mathbf{H} \leftarrow \varnothing$ {set of currently used paths}
  $\mathbf{F} \leftarrow \mathbf{W}$
  $\Delta\mathbf{B} \leftarrow 0$
  $\mathbf{P} \leftarrow \varnothing$ {set of current shortest paths}
  $\mathbf{P} \leftarrow$ **find_shortest_paths**$(\mathbf{F})$ $\{\mathbf{P} = \left\{a_q^k : \min_k \left\{F_q^k\right\}\right\}\}$
  for all $(a_q^k \in \mathbf{P})$ **do**
    $b_q^k \leftarrow d_q$ {assign all the demands to the shortest paths}
  end for
  $\mathbf{H} \leftarrow \mathbf{P}$
  $\mathbf{X} \leftarrow \mathbf{A} \cdot \mathbf{B}$
  $\mathbf{F} \leftarrow$ **update_path_weights**$(\mathbf{X})$
  $\mathbf{P} \leftarrow$ **find_shortest_paths**$(\mathbf{F})$
  $\mathbf{\Delta B} \leftarrow$ **find_shift_direction**$(\mathbf{B}, \mathbf{P}, \mathbf{H}, \mathbf{F})$
  while $(\mathbf{\Delta B} \neq 0)$ **do**
    $\phi \leftarrow$ **find_shift_factor**$(\mathbf{\Delta B}, \mathbf{X})$
    $\mathbf{B} \leftarrow \mathbf{B} + \phi\mathbf{\Delta B}$
    $\mathbf{X} \leftarrow \mathbf{A} \cdot \mathbf{B}$
    $\mathbf{H} \leftarrow \cup_{k=1,\ldots,K_q;q=1,\ldots,Q} \left\{a_q^k : b_q^k > 0\right\}$
    $\mathbf{F} \leftarrow$ **update_path_weights**$(\mathbf{X})$
    $\mathbf{P} \leftarrow$ **find_shortest_paths**$(\mathbf{F})$
    $\mathbf{\Delta B} \leftarrow$ **find_shift_direction**$(\mathbf{B}, \mathbf{P}, \mathbf{H}, \mathbf{F})$
  end while
  if constraints are met **then**
    Return $\mathbf{B}, \mathbf{H}$
  else
    Return INFEASIBLE
  end if

Algorithm 5.5: Pseudo-code for the Modified Flow Deviation Method Applied to the MPSFAP1 Problem

5.5.2. Path Weights Calculation

As with the Tabu Search heuristics, we are interested in redefining the objective function so as to evaluate invalid solutions with higher values than valid solutions. We don't distinguish here among type-1 and type-2 invalid solutions, because we have simplified the constraints into a unique (more restrictive) constraint. The cause of invalidity then is the *overload* of at least one component link of the path being evaluated, in other words, the violation of the link delay constraint θ_i as defined in the previous section. As before, we are also interested in providing a *degree of invalidity* once the constraint is violated, so as to create a slope towards valid path flow vectors (i.e. towards path without overloaded component links). We introduce this evaluation in the way path weights are calculated:

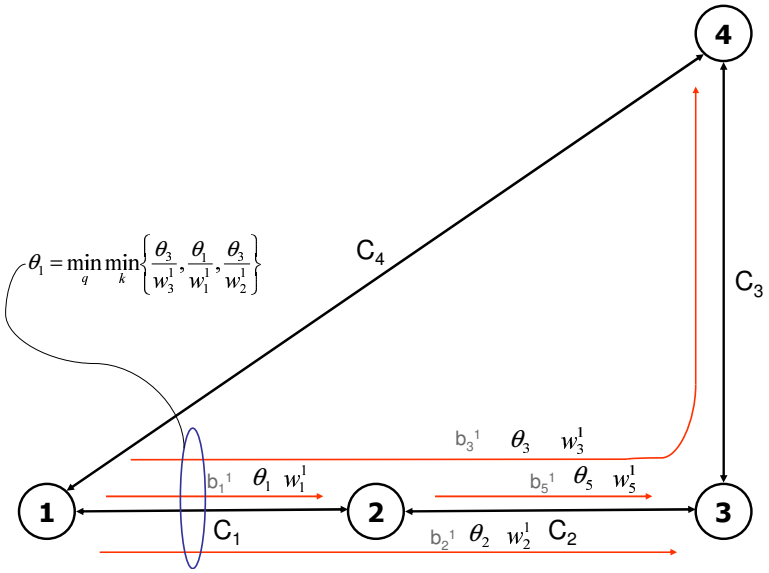


Figure 5.3: Modified Flow Deviation Algorithm: Path Delay Constraint Simplification

$$F_q^k \leftarrow w_q^k + \sum_{i=1}^{M} a_{q,i}^k f(x_i) \tag{5.15}$$

where w_q^k is calculated as in (4.9) and

$$f(x_i) = \begin{cases} 0 & \text{if } x_i \leq C_i - \frac{\lambda}{\theta_i} \\ \frac{x_i}{C_i - \frac{\lambda}{\theta_i}} & \text{if } x_i > C_i - \frac{\lambda}{\theta_i} \end{cases} \tag{5.16}$$

for each link i. The second term in (5.15) adds up an extra weight to the hop-count based path weights for each component link which is overloaded. The value to be added depends on how much flow over the limit is traversing that link, and we have ensured that we add up at least 1 ($x_i > C_i - \frac{\lambda}{\theta_i}$) for every overloaded link in the path. The idea behind this evaluation function is that a path will appear as a shortest path candidate to receive flow from the other paths in the set only if its combination of hop-count and number and proportion of overloaded component links is less than the same combination for all the other paths.

The way we determine the path weights is responsible for making appear more or less candidate paths in a given node pair. The Algorithm 5.5 as it is defined can fail to find a shift direction $\mathbf{\Delta B}$ because no shortest path appears, even when there are unloaded paths in the group that could receive flow. The example of the Figure 5.4 illustrates a case where

an overloaded path is still the shortest path, even when other (longer) unloaded paths are available.

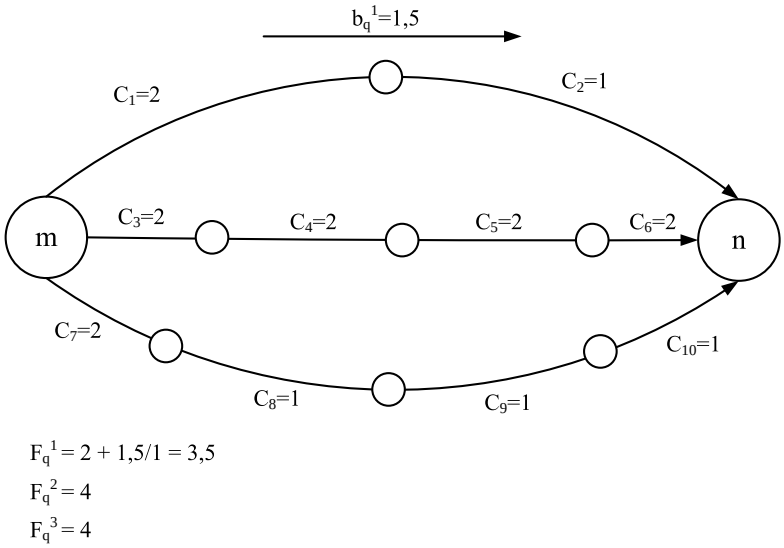


Figure 5.4: MFD Algorithm Example: Overloaded Path is Still the Shortest Path

A way to overcome this problem is to iteratively add up 1 to the weight of each path using at least one overloaded link when $\mathbf{\Delta B} = 0$ and the constraints are still not met, until some $\mathbf{\Delta B} > 0$ is produced up to the diameter of the network. This will make longer unloaded paths to appear as candidates, but only when shorter paths (in terms of hop count) are not available at a given iteration. Algorithm 5.6 improves 5.5 by incorporating the necessary iterations to test for all candidates.

Algorithm 5.7 and 5.8 describe the corresponding procedures to calculate the path weights and to increase the weight of overloaded paths over longest paths to try all possible candidates if necessary.

5.5.3. Determining the Shift Direction

The *shift direction* indicates for which node pair we are going to shift the path flows travelling along non-shortest paths to the shortest path in the sense of F_q^k for the current iteration. In order to determine the shift direction $\mathbf{\Delta B}$ we must consider all candidate shift directions per node pair $\mathbf{\Delta B_q}$ and then chose the one with the higher benefit in terms of reducing the total overload in the network with respect to the cost of the newly introduced path. The shift direction for each node pair q is calculated as:

```
⋮
while (ΔB ≠ 0) do
  φ ← find_shift_factor(ΔB, X)
  ⋮
  ΔB ← find_shift_direction(B, P, H, F)
  if (ΔB = 0) and (constraints are not met) then
    threshold ← 1
    F_b ← F
    while (ΔB = 0) and (threshold < network_diameter) do
      F_b ← increase_threshold_level(F_b, threshold)
      P ← find_shortest_paths(F_b)
      ΔB ← find_shift_direction(B, P, H, F_b)
      threshold ← treshold + 1
    end while
  end if
end while
if constraints are met then
  Return B, H
else
  Return INFEASIBLE
end if
```

Algorithm 5.6: Pseudo-code for the Improvement to the MFD Method to Consider All Candidate Paths

```
Require: X {the link flows vector}
Ensure: update_path_weights(X)
  for q = 1 to Q do
    for k = 1 to K_q do
      F_q^k ← w_q^k
      for i = 1 to M do
        if (a_{q,i}^k = 1) and (x_i > C_i − λ/θ_i) then
          F_q^k ← F_q^k + x_i / (C_i − λ/θ_i)
        end if
      end for
    end for
  end for
  Return F
```

Algorithm 5.7: Pseudo-code for Calculating the Path Weights

$$\mathbf{\Delta B_q} = \left\{\Delta b_q^k\right\} = \begin{cases} d_q - b_q^k & \text{if } a_q^k \in \mathbf{P} \\ -b_q^k & \text{if } a_q^k \in \mathbf{H} \wedge a_q^k \notin \mathbf{P} \\ 0 & \text{otherwise} \end{cases} \tag{5.17}$$

for $k = 1, \ldots, K_q$ and all node pairs. $\mathbf{\Delta B_q}$ expresses the fact that we allow for potentially

Require: $\mathbf{F}, treshold$
Ensure: **increase_treshold_level**($\mathbf{F}, treshold$)
  $overloaded \leftarrow$ NO
  for $q = 1$ to Q **do**
    for $k = 1$ to K_q **do**
      for $i = 1$ to M **do**
        if $(a_{q,i}^k = 1)$ and $(x_i > C_i - \frac{\lambda}{\theta_i})$ **then**
          $overloaded \leftarrow$ YES
        end if
      end for
      if $(overloaded =$ YES$)$ **then**
        $F_q^k \leftarrow F_q^k + 1$
      end if
    end for
  end for
  Return **F**

Algorithm 5.8: Pseudo-code for Increasing the Path Weights for Overloaded Paths

all the demand d_q to be shifted to the newly found shortest path (we also consider the case where the path was already carrying some traffic), and all currently transported flow to leave a non-shortest path. The set **P** contains the shortest paths for the current iteration, and the set **H** all the currently used paths. Algorithm 5.9 shows the pseudo-code to calculate the *shift direction*.

5.5.4. Determining the Shift Factor

Each shift direction in the paths for a given node pair q will produce a shift on the component links due to that node pair. The corresponding shifts $\mathbf{\Delta X_q}$ in the links are calculated from the path flow shifts for the node pair q as:

$$\Delta x_{i,q} = \sum_{k=1}^{K_q} a_{q,i}^k \Delta b_q^k \tag{5.18}$$

From the $\Delta x_{i,q}$ we can determine the proportion of flow $\phi_{i,q}$ that can be moved for a given node pair q on each link i as:

$$\phi_{i,q} = \begin{cases} \frac{x_i}{|\Delta x_{i,q}|} & \text{if } \Delta x_{i,q} < 0 \\ \frac{C_i - \frac{\lambda}{\theta_i} - x_i}{\Delta x_{i,q}} & \text{if } \Delta x_{i,q} > 0 \wedge x_i < C_i - \frac{\lambda}{\theta_i} \\ 0 & \text{otherwise} \end{cases} \tag{5.19}$$

Require: $\mathbf{B}, \mathbf{P}, \mathbf{H}, \mathbf{F}$
Ensure: **find_shift_direction**$(\mathbf{B}, \mathbf{P}, \mathbf{H}, \mathbf{F})$
  $\mathbf{\Delta B} \leftarrow 0$
  $\mathbf{\Delta B_q} \leftarrow 0$
  $\psi_q \leftarrow 0$
  $\psi_{BEST} \leftarrow 0$
  $\mathbf{X} \leftarrow \mathbf{A} \cdot \mathbf{B}$
  for $q = 1$ to Q **do**
    for $k = 1$ to K_q **do**
      if $(a_q^k \in \mathbf{P})$ **then**
        $\Delta b_q^k \leftarrow d_q - b_q^k$
      else if $(a_q^k \in \mathbf{H})$ **then**
        $\Delta b_q^k \leftarrow -b_q^k$
      end if
    end for
    $\psi_q \leftarrow$ **calculate_choice_index**$(\mathbf{\Delta B_q}, \mathbf{X}, \mathbf{H}, \mathbf{P}, \mathbf{F})$
    if $(\psi_q > \psi_{BEST})$ **then**
      $\mathbf{\Delta B} \leftarrow \mathbf{\Delta B_q}$
      $\psi_{BEST} \leftarrow \psi_q$
    end if
  end for
  Return $\mathbf{\Delta B}$

Algorithm 5.9: Pseudo-code for Calculating the Shift Direction

When $\Delta x_{i,q} < 0$, we are actually shifting flow out of link i, the expression (5.19) indicates that we can at most move out the total flow x_i allocated to it. On the other hand, if $\Delta x_{i,q} > 0$ we are trying to move flow into link i. In that case, expression (5.19) says that if the link is not overloaded, we can shift flow in up to the constraint for that link, given by $C_i - \frac{\lambda}{\theta_i}$. Having calculated all the shift factors for each link, we take the *shift factor* for the node pair to be the minimum for all the links:

$$\phi_q = \min_i \{\phi_{i,q} : \phi_{i,q} > 0\} \tag{5.20}$$

Finally, if $\phi_q > 1$, then we need to set $\phi_q = 1$ (i.e. the maximum flow that can be shifted out from a link is the flow already traversing it. Conversely, we cannot shift in to a link more flow than $\Delta x_{i,q}$, even if there is still available capacity.

5.5.5. Determining the Best Node Pair

The algorithm 5.9 determines the shift direction by choosing among the shift directions for each node pair, the one with the best *choice index* ψ_q. We said that the best shift direction would be the one producing the higher benefit in terms of reducing the total overload in the network with respect to the cost of the newly introduced path.

Require: $\mathbf{\Delta B}, \mathbf{X}$ {called with $\mathbf{\Delta B}$ calculates ϕ, called with $\mathbf{\Delta B_q}$ calculates ϕ_q}
Ensure: **find_shift_factor**$(\mathbf{\Delta B}, \mathbf{X})$
  $\phi_q \leftarrow 0$
  $\phi_{i,q} \leftarrow 0$
  for $q = 1$ to Q **do**
    $\mathbf{\Delta X_q} \leftarrow \mathbf{A} \cdot \mathbf{\Delta B_q}$
    for $i = 1$ to M **do**
      if $(\Delta x_{i,q} < 0)$ **then**
        $\phi_{i,q} \leftarrow \frac{x_i}{|\Delta x_{i,q}|}$
      else if $(\Delta x_{i,q} > 0)$ and $(x_i < C_i - \frac{\lambda}{\theta_i})$ **then**
        $\phi_{i,q} \leftarrow \frac{C_i - \frac{\lambda}{\theta_i} - x_i}{\Delta x_{i,q}}$
      end if
      if $(\phi_{i,q} > 0)$ and $(\phi_{i,q} < \phi_q)$ **then**
        $\phi_q \leftarrow \phi_{i,q}$
      end if
    end for
  end for
  if $(\phi_q > 1)$ **then**
    $\phi_q \leftarrow 1$
  end if
  Return ϕ_q

Algorithm 5.10: Pseudo-code for Calculating the Shift Factor

We calculate then a choice index for every pair of nodes as follows:

$$\psi_q = \begin{cases} \frac{\phi_q \sum_{i=1}^{M} \gamma_i |\Delta x_{i,q}|}{\widetilde{F}_q} & \text{if } \gamma_i = 1 \text{ for some } i \\ \frac{|\Delta x_{i,q}|}{\widetilde{F}_q} & \text{if } \gamma_i = 0 \text{ for all } i \wedge \phi_q \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.21}$$

where $\gamma_i = 0$ if link i is not overloaded $(x_i > C_i - \frac{\lambda}{\theta_i})$ and 1 if the link is overloaded. $\widetilde{F}_q = \sum_{k=1}^{K_q} h_q^k F_q^k$ (i.e. the sum of all used paths in the current iteration). ψ_q is actually measuring the amount of reduction on the overloaded links relative to the cost of the set of used paths for a given node pair q, including the paths added in the current iteration (if some path is added). If there are no overloaded links, we allow the algorithm to try and rearrange the path flows to see if there is any place for flow aggregation. We add this possibility in the second option in (5.21): flow aggregation will only be possible if the *shift factor* for the candidate node pair is 1 or greater, since the flow would be further splitted otherwise.

Finally, the MFD algorithm can be also adapted to solve the MPSFAP2 problems. Simply changing the evaluation of the weights for each path during the algorithm to:

Require: $\mathbf{\Delta B_q}, \mathbf{X}, \mathbf{H}, \mathbf{P}, \mathbf{F}$
Ensure: **calculate_choice_index**$(\mathbf{\Delta B_q}, \mathbf{X}, \mathbf{H}, \mathbf{P}, \mathbf{F})$
  $\psi_{q1}, \psi_{q2}, \psi_q, \psi_{BEST} \leftarrow 0$
  $\phi_q \leftarrow 0$
  $overload \leftarrow$ NO
  for $q = 1$ to Q **do**
    $\phi_q \leftarrow$ **find_shift_factor**$(\mathbf{\Delta B_q}, \mathbf{X})$
    for $i = 1$ to M **do**
      $\mathbf{\Delta X_q} \leftarrow \mathbf{A} \cdot \mathbf{\Delta B_q}$
      if $(x_i > C_i - \frac{\lambda}{\theta_i})$ **then**
        $\psi_{q1} \leftarrow \psi_{q1} + \phi_q |\Delta x_{i,q}|$
        $overload \leftarrow$ YES
      else if $(\phi_q \geq 1)$ and $(|\Delta x_{i,q}| > 0)$ **then**
        $\psi_{q2} \leftarrow |\Delta x_{i,q}|$
      end if
    end for
    if $(overloaded =$ YES$)$ **then**
      $\psi_q \leftarrow \psi_{q1}$
    else
      $\psi_q \leftarrow \psi_{q2}$
    end if
    for all $a_q^k \in \mathbf{H} \cup \mathbf{P}$ **do**
      $\widetilde{F}_q \leftarrow \widetilde{F}_q + F_q^k$
    end for
    $\psi_q \leftarrow \frac{\psi_q}{\widetilde{F}_q}$
    if $(\psi_q > \psi_{BEST})$ **then**
      $\psi_{BEST} \leftarrow \psi_q$
    end if
  end for
  Return ψ_{BEST}

Algorithm 5.11: Pseudo-code for Calculating the Choice Index

$$F_q^k = w_q^k + h_q^k \sum_{i=1}^{M} a_{q,i}^k \frac{\lambda}{C_i - x_i} + \sum_{i=1}^{M} a_{q,i}^k f(x_i) \tag{5.22}$$

with $f(x_i)$ defined as in (5.15). This way of calculating the path weights consider the path delay as part of its length. Since only the equation (5.15) has been considered in our implementation for calculating the path weights, we restrain ourselves to the an analysis of the results obtained from the MFD algorithm for the MPSFAP1 problems.

5.5.6. Evaluation of the MFD Algorithm for MPSFAP

As with the TS heuristics in Section 5.3.4, we run the MFD algorithm for the same set of 25 matrixes, and network topologies NET1 and NET2. We then compare the results obtained

from MFD to the results obtained from the deterministic solver MINLP. Since the MFD algorithm runs up to wether obtaining a feasible solution (which is also a minimum cost approximation to the optimal) or trying all possible candidate shortest paths, the number of iterations is not a parameter as in TS. Recalling the definition of $f(s)\%_{TS}$ given in (5.10), we define here $f(s)\%_{MFD}$ as follows:

$$f(s)\%_{MFD} = \frac{f(s)_{MFD} - f(s)_{MINLP}}{f(s)_{MINLP}} 100 \tag{5.23}$$

Recalling the concepts used when working with the TS algorithm, s is a solution as defined by (5.1). The *path flow vector* $\mathbf{B}$ and the *solution* s are used here as synonymous. Table 5.4 shows the average values of the objective function for the set of demand matrixes considered. It must also be noted (as is the case with the TS algorithm) that the evaluation function $f(s) = \sum_{q=1}^{Q} \sum_{k=1}^{K_q} h_q^k F_q^k$ become the objective function given by (4.6), with w_q^k calculated as in (4.9) when the problem is feasible. This allows us to compare the values of the objective functions obtained from the three methods directly for feasible problems.

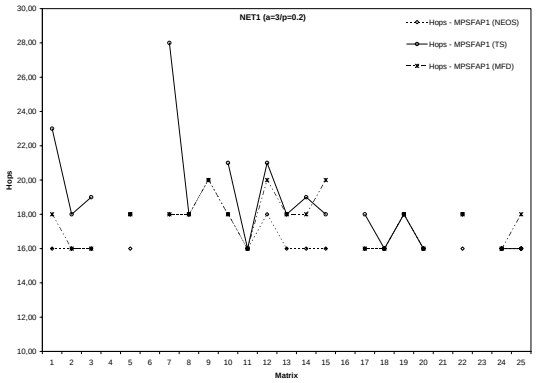
		MPSFAP1				
		$f(s)_{MINLP}$	$f(s)_{TS}$	$f(s)_{MFD}$	$f(s)\%_{TS}$	$f(s)\%_{MFD}$
NET1	$a = 3/p = 0.2$	16.70	18.68	17.60	11.86	5.39
	$a = 4/p = 0.2$	16.00	17.20	16.24	7.50	1.50
NET2	$a = 5/p = 0.2$	15.16	16.74	15.78	8.18	4.09
	$a = 6/p = 0.2$	14.60	15.88	15.04	8.77	3.01

Table 5.4: Quality of the Solutions from MFD with respect to TS and MINLP for MPSFAP1

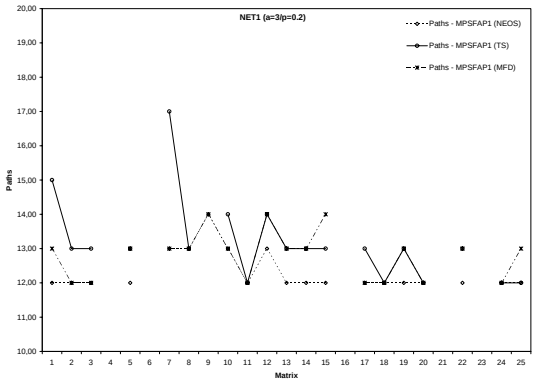
We can easily see from the values of $f(s)\%_{MFD}$ that the approximation to the optimal values of $f(s)$ obtained from the MINLP solver are better in all cases than the approximation obtained from tabu search heuristics. We have included the values relative to TS in all tables and figures for direct comparison. Again, we see that the approximation improves when the load increases for a given topology, and also improves when the size of the problem increases (i.e. when the network gets more densely connected or the number of nodes increases).

		MPSFAP1					
		Hops			**Paths**		
		MINLP	**TS**	**MFD**	**MINLP**	**TS**	**MFD**
NET1	$a = 3/p = 0.2$	16.70	18.68	17.60	12.30	13.16	12.80
	$a = 4/p = 0.2$	16.00	17.20	16.24	12.00	12.52	12.12
NET2	$a = 5/p = 0.2$	15.16	16.74	15.78	12.37	12.52	12.12
	$a = 6/p = 0.2$	14.60	15.88	15.04	12.12	12.84	12.42

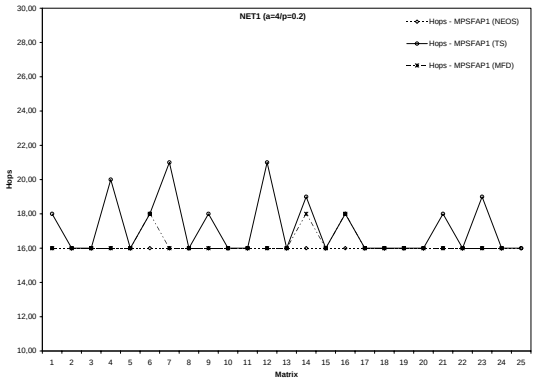
Table 5.5: Quantity of Paths and Hops for the Solutions from MINLP, TS and MFD for MPSFAP1



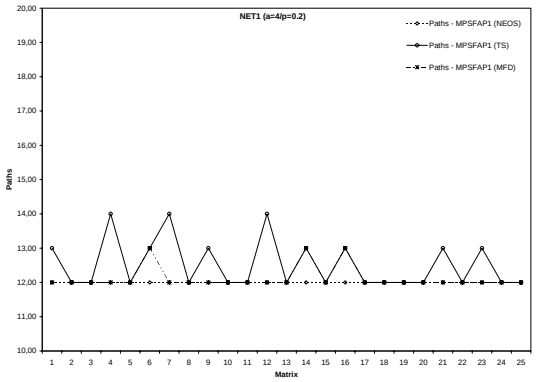
(a) Hops, $a = 3$



(b) Paths, $a = 3$



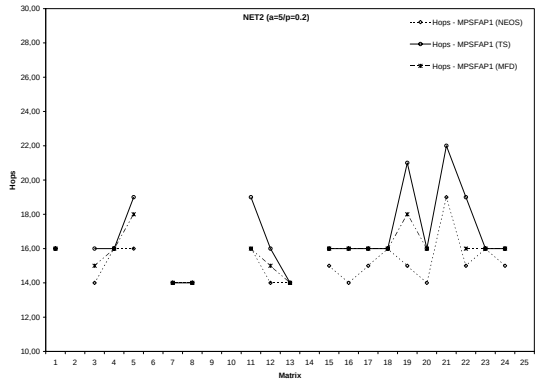
(c) Hops, $a = 4$



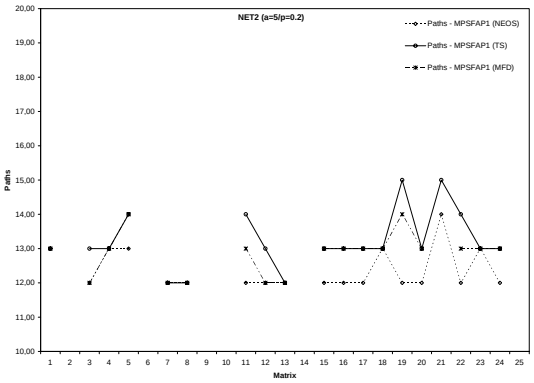
(d) Paths, $a = 4$

Figure 5.5: Results from MINLP Solver (NEOS), Tabu Search (TS) and Modified Flow Deviation (MFD) for NET1

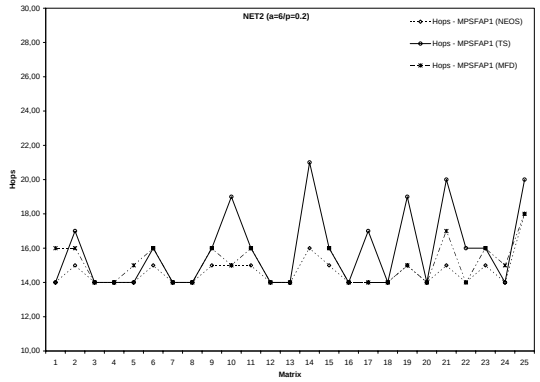
Here again, as the problem size increases, a larger number of node pairs will appear as candidates for becoming a *shift direction*, as there will be more paths available connecting each node pair and more node pairs to consider. It can be observed as well that, compared to the values obtained from MINLP solver for MTDFAP problem (the one minimizing the total delay of used paths in the network), the MFD algorithm always constitutes an improvement on the proposed objectives: to obtain layouts minimizing the total number of hops (and indirectly of paths) in the layout. Figures 5.5 and 5.6 show the number of hops and paths obtained from the three methods for NET1 and NET2 respectively. For almost all matrixes, the TS algorithm constitutes an *upper bound* for the number of hops and paths obtained from the MFD algorithm. Finally, CPU times required by MFD under the same conditions than TS are under the second for both topologies, already ten times less than times consumed by TS.



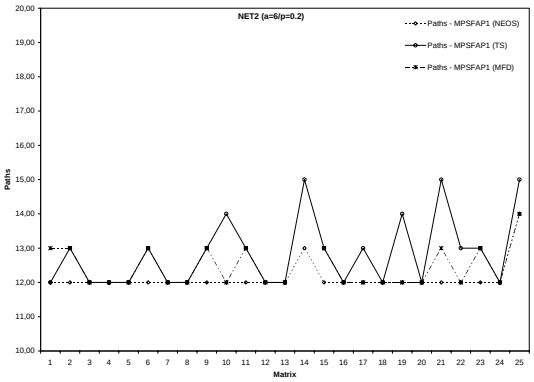
(a) Hops, $a = 5$



(b) Paths, $a = 5$



(c) Hops, $a = 6$



(d) Paths, $a = 6$

Figure 5.6: Results from MINLP Solver (NEOS), Tabu Search (TS) and Modified Flow Deviation (MFD) for NET2

In next Chapter we compare results obtained from both MFD and TS for two large topologies obtained from real-life size networks. This results allow us to analyze the behavior of the described heuristics as well as the pertinence and applicability of each one to day-to-day operational environment.

5.6. Conclusions

In the previous Chapter we have shown the interest, from an operator's standpoint, of considering complexity objectives when designing optimal MPLS layouts. In the present Chapter the major contribution consist in the definition and implementation of two heuristic methods to deal with the MPSFAP problems. The first heuristic is a *generic heuristic* using Tabu Search (TS) methods and particular evaluation of the *generated* solution to explore the solution space. The second heuristic is an *ad-hoc heuristic* which makes use of

the particular structure and nature of the problem to intelligently search the solution space rather than randomly. This last heuristics is based on a modification to the general procedure of the well known Flow Deviation Method to solve the multicommodity flow problem for convex objective functions. The resulting algorithm is referred as MFD (Modified Flow Deviation) algorithm. Results obtained with both heuristic methods are compared to the solutions obtained using the deterministic solver available for the MINLP problems, in order to establish the quality of the solutions obtained through heuristics methods with respect to the optimal solution. Due to problem size limitations when using the deterministic solvers, results are compared for small topologies in order to obtain some insight in the behavior of the heuristics. Results for the MTDFAP problem are also given as a reference. Finally results obtained with the TS and the MFD heuristics are compared one to another in order to establish the relative quality of the solutions obtained through both methods.

Results show that both, TS and MFD yield fairly good approximations for small network topologies. MFD however, shows better quality results with respect to those obtained by using the TS algorithm. Besides, the execution times on a standard personal computer are far lower for MFD than for TS. The figures obtained for small networks encourage us to continue further exploring both heuristic methods, which we do in the next Chapter. Although small network topologies do not offer much space for a real difference among the various optimization criteria, we can already see that we obtain layouts for the MPSFAP objective functions using less hops and consequently less paths than those obtained for MTDFAP objectives (i.e. the classic performance objectives used extensively through the literature). In next Chapter we evaluate both heuristic methods for two large network topologies, and we explore the results in order to obtain more insight in the behavior of the cost functions proposed, as well as to evaluate the performance of the heuristic methods developed.

6. Results and Analysis of the MPSFAP Problems for Large Networks Using Tabu Search (TS) and Modified Flow Deviation (MFD) Methods

In the previous Chapter, we have developed two heuristic methods to solve the MPSFAP problem: a Tabu Search meta-heuristic algorithm based on meta-heuristics, and a Modified Flow Deviation algorithm based on the Frank-Wolfe method (ad-hoc heuristics). These algorithms are expected to allow us to overcome the problem size limitations described in Chapter 4 when evaluating and validating the proposed model for MPLS layout complexity optimization through deterministic solvers. In the mentioned Chapter, we have individually evaluated each one of the developed heuristics for small network topologies like the ones used to first evaluate the models in Chapter 4. We have concluded that both heuristics are fairly good approximations of the optimal values obtained through the deterministic MINLP solver. We have also noted that the MFD algorithm outperforms the TS algorithms both in terms of computational effort and in terms of the quality of the results for the studied topologies.

In the present Chapter, we extend the study of both heuristics to large network topologies. Two network topologies are presented, representing national span networks from Europe (France) and United States. Then results for different load situations using both algorithms for the presented network topologies are obtained and compared. Finally, based on the result comparison between the two heuristics, further analysis on results for the MPSFAP problem using the MFD algorithm are presented.

6.1. Considered Network Topologies

We run the TS and MFD algorithms on two medium-sized to large topologies issued from real-life networks, covering a national span, both in Europe and United States. The VTHD network shown in Figure 6.1 is modelled after the French National Research Network, deployed in the context of the VTHD (*Vraiment Très Haut Débit*)[63] research project, where only the core nodes of network are represented. In a similar way, the NSF network shown in Figure 6.2 is a model of the Internet in the US at the time of ARPANET. We have chosen these two topologies because they represent large networks issued from real-life planning, with medium to high degrees of connectivity (2.66 for VTHD and 3 for NSF) transporting traffic aggregates.

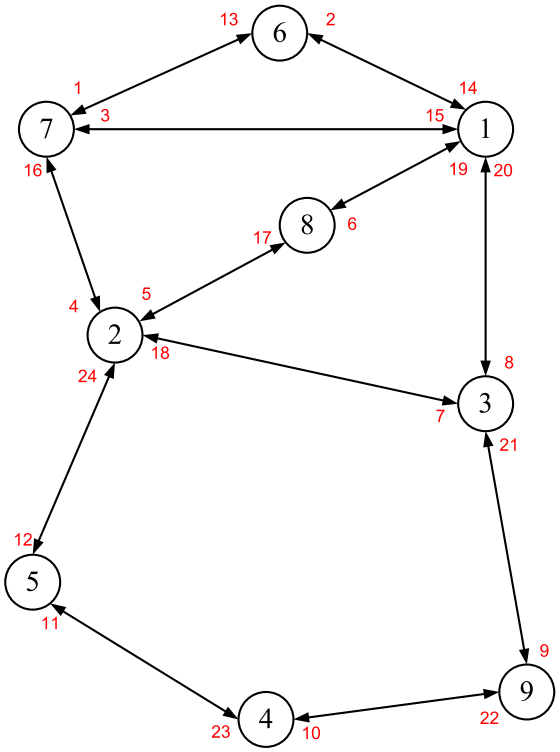


Figure 6.1: VTHD Network Topology

The VTHD topology is a 9 nodes and 24 links network, producing a total of 550 paths connecting all node pairs (i.e. the arc-path incidence matrix is a $K \times M = 550 \times 24$ matrix). NSF topology is a 14 nodes and 42 links network, producing a total of $14,250$ paths. All links are set to a Capacity of 2.5 Gbps, corresponding to a single WDM wavelength.

6.2. Comparison from TS and MFD for the MPSFAP on Large Networks

A set of 25 demand matrixes for two load levels is solved for each topology using both TS and MFD algorithms. We chose the parameter a when generating the traffic matrixes for each topology so as to produce some infeasible problems in high load and none in medium load. The chosen parameters to generate the traffic matrixes are then $a = 115$ and $a = 120$ corresponding to high and medium load for VTHD topology; correspondingly we chose $a = 37$ and $a = 40$ for NSF network. The parameter p representing the asymmetry for some node pairs is set to 0.2 in all cases.

It is important to note that the algorithmic implementation of the TS method is highly

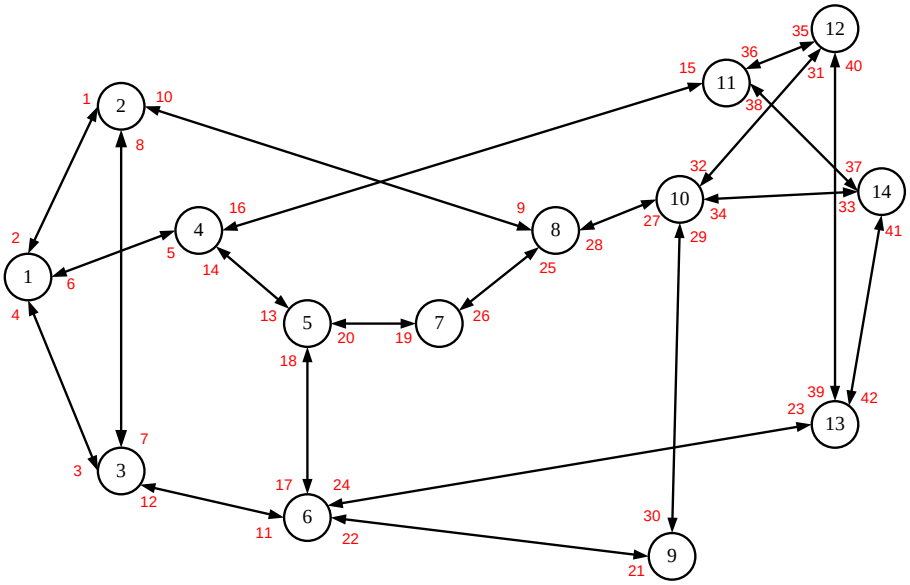


Figure 6.2: NSF Network Topology

memory consuming and computationally intensive. A number of neighbors need to be generated and kept in memory at each iteration, limiting the maximum size of the problems that can be given as input (although much larger than for deterministic solvers). At the same time, a second limitation appears when increasing the number of paths for the same topology. Indeed, the time consumed to accomplish each iteration of the TS algorithm depends directly on the total number of paths in **A** and the number of neighbors to generate, constituting a practical limitation when running large sets of demand matrixes. Reducing the number of neighbors to generate at each iteration of the the TS algorithm reduces the probability of finding a good neighbor, and as such, the quality of the progression in the solution space. An alternative to reducing the number of neighbors is to reduce the number of paths considered in the arc-path incidence matrix **A**. Observing from results in previous chapters, the path multiplicity of the solutions (even for high load levels) is near unity. We observe as well that, as the path weights in the objective function for MPSFAP1 are the hop counts, the paths used in the solution tend to be the shortest or next to shortest path connecting each node pair. Providing then a reduced set of paths to the algorithm would not greatly penalize the quality of the solution if enough alternative paths (chosen in increasing lengths) connecting each node pair are provided. Consequently, we consider a reduced arc-path incidence matrix, containing a total of 910 paths (i.e. a set of 5 paths for every node pair), as a good trade-off between path candidates and time consumed. It is also important to note that the algorithmic implementation of the MFD algorithm doesn't present this limitation. In fact, only one instance of the problem is kept in memory at each

iteration, and the time consumed to perform each iteration does not depend so strongly on the total number of paths. However, the same reduced arc-path incidence matrix is used as input for both methods in order to perform result comparisons under the same conditions.

The Tabu Search algorithm is asked in all cases to generate 50 neighbors at each iteration. It runs for a total of 10 iterations, and it stops if the value of the objective function does not change for the last 5 iterations (*inefficient* iterations). The Modified Flow Deviation algorithm runs until a feasible solution is found or until no shift direction can be found in the sense of improving the value of the objective function. Table 6.1 shows the time consumed in average by each algorithm to obtain solutions for the 25 matrix set on a Intel Pentium IV© running at 2.4 GHz with 512 Mb RAM. It also shows the feasibility ratio for each run. In all cases, the end-to-end path delay constraint is set to $\theta_q = 50\mu sec$.

MPSFAP1 25 matrix set		**Tabu Search**		**Modified Flow Deviation**	
		Ex.Time (*sec*)	**Feas. (%)**	**Ex.Time (*sec*)**	**Feas. (%)**
VTHD	$a = 115$	1380	64	0.1	60
	$a = 120$	1380	100	0.1	84
NSF	$a = 37$	10020	72	1.5	60
	$a = 40$	10020	96	1.5	92

Table 6.1: Average Execution Times and Feasibility Ratios for MPSFAP1 by Using TS and MFD Algorithms

Table 6.2 shows the average value of the objective function for MPSFAP1 (i.e. total number of hops) obtained with both TS and MFD. In order to compare the quality of the results, we redefine (5.10) and (5.23) as follows:

$$f(s)\%_{MFD/TS} = \frac{f(s)_{TS} - f(s)_{MFD}}{f(s)_{MFD}} 100 \tag{6.1}$$

MPSFAP1		$f(s)_{TS}$	$f(s)_{MFD}$	$f(s)\%_{MFD/TS}$
VTHD	$a = 115$	160.94	142.60	12.86
	$a = 120$	150.04	140.81	6.55
NSF	$a = 37$	422.11	397.13	6.29
	$a = 40$	405.71	392.91	3.26

Table 6.2: Average Values of the Objective Function for MPSFAP1 by Using TS and MFD Algorithms

From Table 6.1 it can be seen that the MFD algorithm largely outperforms the TS algorithm. Execution times for the MFD algorithm are significatively smaller than execution times for the TS algorithm (in the order of 10^4 times), while the quality of the results are

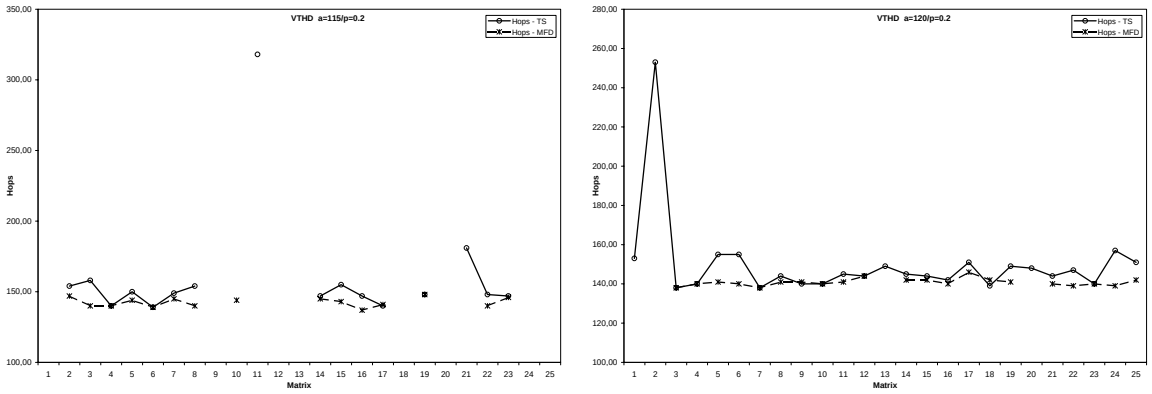
significatively better for the MFD algorithm in terms of value of the objective function under all load conditions, as it can be observed in the Table 6.2. However, the feasibility ratio is better for the TS algorithm than for the MFD algorithm. This is due mainly to the end-to-end path constraint simplification, which imposes more restrictive constraints to the link loads. Table 6.3 shows the average quantity of paths and the average maximum end-to-end path delay (τ_q^k) obtained for MPSFAP1 using TS and MFD algorithms on VTHD and NSF topologies for the two load situations.

MPSFAP1 25 matrix set		**Tabu Search**		**Modified Flow Deviation**	
		Paths	Max. τ_q^k (μsec)	**Paths**	Max. τ_q^k (μsec)
VTHD	$a = 115$	75.38	26.05	72.70	15.89
	$a = 120$	73.28	21.83	71.76	14.27
NSF	$a = 37$	189.06	39.66	183.40	25.65
	$a = 40$	185.46	34.33	182.30	19.23

Table 6.3: Average Paths and Average Maximum End-to-end Path Delay for MPSFAP1 by Using TS and MFD Algorithms

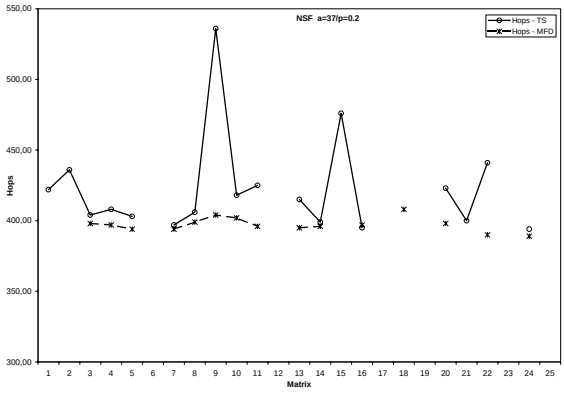
Due to the simplification of the end-to-end path delay constraints, more restrictive constraints are imposed on link loads, resulting in a higher infeasibility ratio. In fact, as we can see in Table 6.3 and Figure 6.5, the end-to-end path delay on layouts calculated with MFD algorithm are significantly lower than those on layouts calculated with the TS method. More, we verify that resulting path delays with MFD runs are far from the imposed constraint θ_q. Increasing then the end-to-end path delay constraints for MFD would allow for higher feasibility rates without violating the originally set constraints. This possibility is further explored in next section.

Figures 6.3 and 6.4 show the detailed hop count (the value of the objective function) and the quantity of paths used by each layout obtained as a solution by means of TS and MFD to the corresponding demand matrix. We see that in all cases the MFD algorithm obtains layouts with a lower number of hops and paths than those obtained by the TS algorithm. To increase the quality of the solutions obtained with TS we should increase the number of iterations over which the algorithm explores the solution space, further increasing the necessary CPU time. The fact that CPU times required by TS to obtain solutions of comparable but lower quality than those obtained with MFD are measured in *hours* instead of in seconds or minutes, tells us that the algorithm is not compatible with operational times. In fact, if our objective is to adapt the network to changing traffic conditions on a long-term basis, we should be able to recalculate the layout in less time than the reconfiguration cycle. Having found that the MFD algorithm performs better quality calculations in lower CPU times (several orders of magnitude lower), further development of the TS algorithm is no

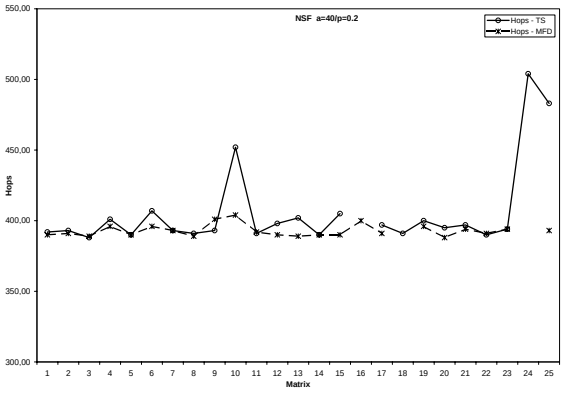


(a) VTHD, $a = 115$

(b) VTHD, $a = 120$



(c) NSF, $a = 37$

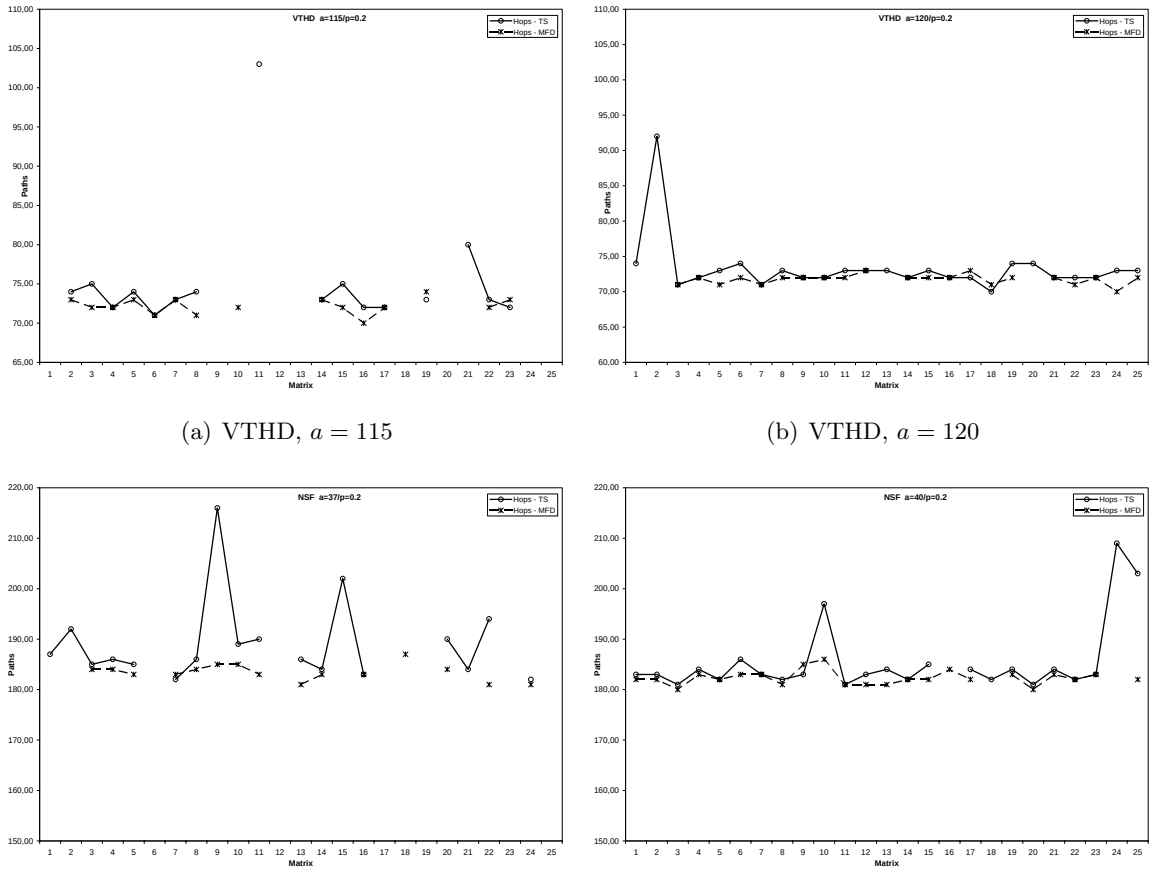


(d) NSF, $a = 40$

Figure 6.3: Value of The Objective Function for MPSFAP1 (Total Hops) for VTHD and NSF Network Topologies using TS and MFD Algorithms

longer justified for solving the reconfiguration problem defined in next Chapter.

Regarding the behavior of the objective function we note that, as it is expected, MPSFAP1 requires more paths (and in consequence more hops) under heavy load conditions than it does under medium load conditions. Indeed, as the demand levels increase, the traffic need to be splitted among the available paths in order to be able to absorb the demand. Further, it can be observed that MPSFAP1 objective keeps a low number of required paths overall. Since VTHD topology is a 9 node network, the total number of node pairs is $N(N-1) = 72$, while for NSF the total is 182 pairs. We see from Table 6.3 that the average required number of paths for VTHD by MFD algorithm is 72.70 under heavy load and 71.76 under light load. This means that in average, solutions obtained for MPSFAP1 for VTHD are using less than 1 extra path under heavy load to transport the whole demand (under medium load the average number of required paths is less than 72 because on some demand matrixes there



(a) VTHD, $a = 115$

(b) VTHD, $a = 120$

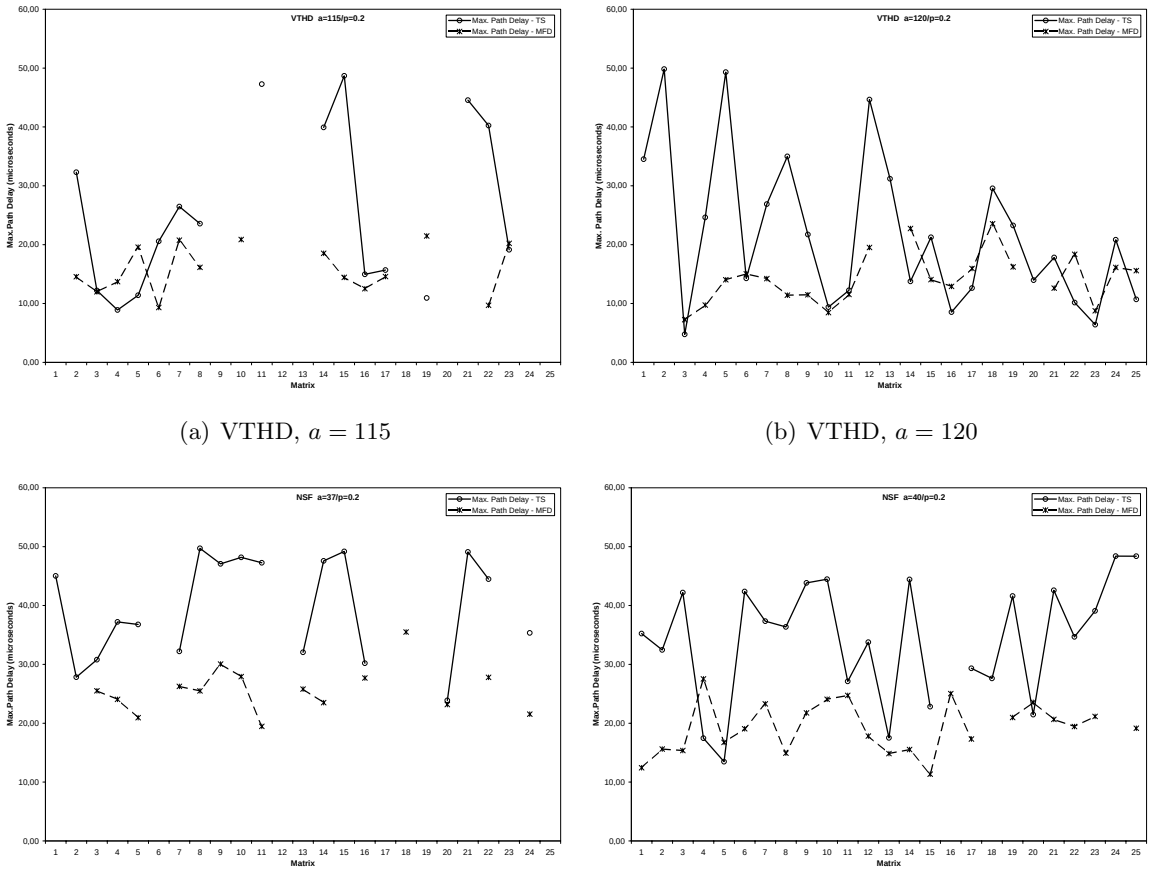
(c) NSF, $a = 37$

(d) NSF, $a = 40$

Figure 6.4: Used Paths for Layouts Obtained from MPSFAP1 for VTHD and NSF Network Topologies using TS and MFD Algorithms

are nodes with 0 demand). In the case of NSF, less than 2 extra paths were required to route the whole demand under heavy load, while less than 1 extra path is necessary under medium load. The average hop-count for VTHD is 1.96, while for NSF is 2.16. These figures are close to the single-path routing case, leading us to think that the optimal layouts according to MPSFAP1 could be built using an IGP protocol in a pure IP environment. However, to find a unique IGP metric compatible with the optimal general-routing problem (studied in [40]) seems particularly difficult. Alternatives using a mix of IGP metrics for the compatible part of the optimal layout and complementary LSPs to complete it have been proposed.

Finally, paying attention to the individual maximum end-to-end path delay in Figure 6.5, we see that they do not largely depend on the load level. This is explained by the fact that, as the MPSFAP1 objective is to minimize the number of hops (consequently of paths) used in the layout, as far as there is some room in the links, the traffic will be aggregated up



(a) VTHD, $a = 115$

(b) VTHD, $a = 120$

(c) NSF, $a = 37$

(d) NSF, $a = 40$

Figure 6.5: Maximum Path Delay in Layouts Obtained from MPSFAP1 for VTHD and NSF Network Topologies using TS and MFD Algorithms

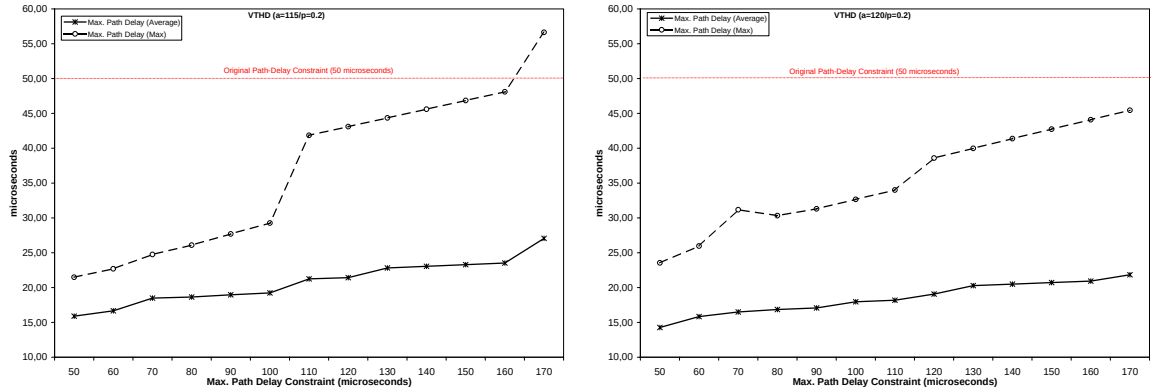
to the limit imposed by the capacity and path delay constraints. As the results show, the minimum set of paths is obtained to the expense of growing end-to-end path delays, proving that the introduced constraints are key to keeping the intended QoS guarantees.

In next section we further analyze results for the VTHD and NSF topologies using the MFD algorithm, as the running times and quality of the obtained results as shown in the present section allow for extensive tests to gain insight in the behavior of the dimensioning problem on large networks.

6.3. Result Analysis of Large Networks by Using MFD

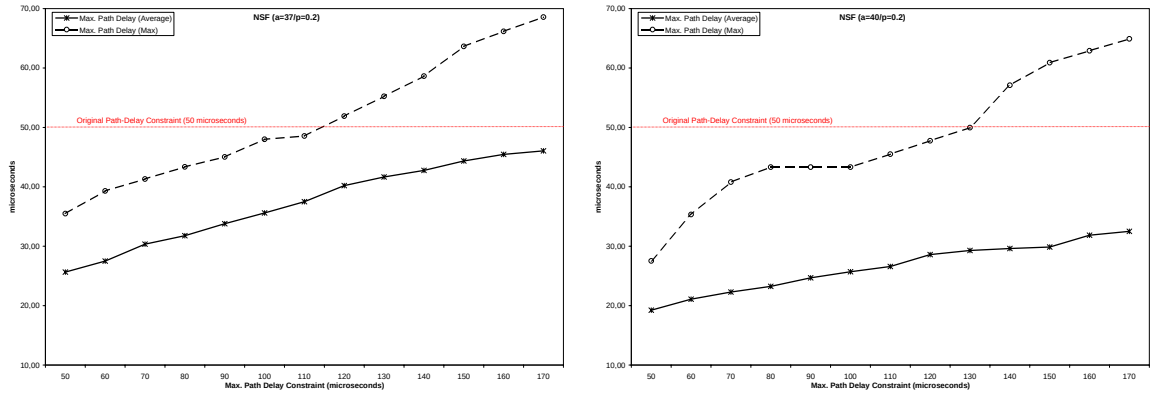
Given the observed performance of the MFD algorithm as established in previous section, we further analyze some results obtained for the VTHD and NSF networks. For the two

values of the parameter a in the traffic matrixes, corresponding to high and medium load levels for each network, we explore results obtained when varying the end-to-end path delay constraint θ_q for every node pair. The tightest value is set to 50 $\mu seconds$, and the constraint is relaxed up to 170 $\mu seconds$. The feasibility ratio, and the evolution of the observed number of hops and paths, as well as the evolution of the average maximum and absolute maximum end-to-end path delay with respect to the changes in the constraints are analyzed. For simplicity, the path delay constraints are all set to the same value.



(a) VTHD, $a = 115$ (b) VTHD, $a = 120$

Figure 6.6: Maximum Path Delay and Average Maximum Path Delay for VTHD Network Topologies Using MFD Algorithm, as a Function of Delay Constraint

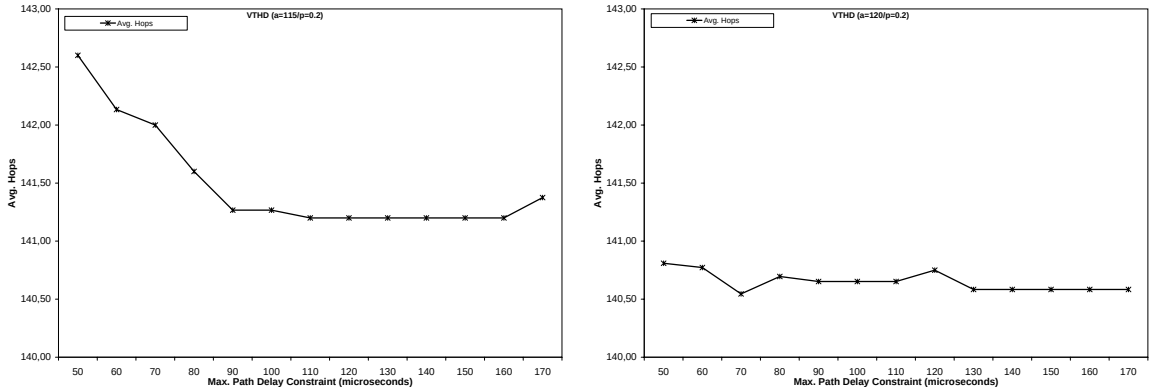


(a) NSF, $a = 37$ (b) NSF, $a = 40$

Figure 6.7: Maximum Path Delay and Average Maximum Path Delay for NSF Network Topologies Using MFD Algorithm, as a Function of Delay Constraint

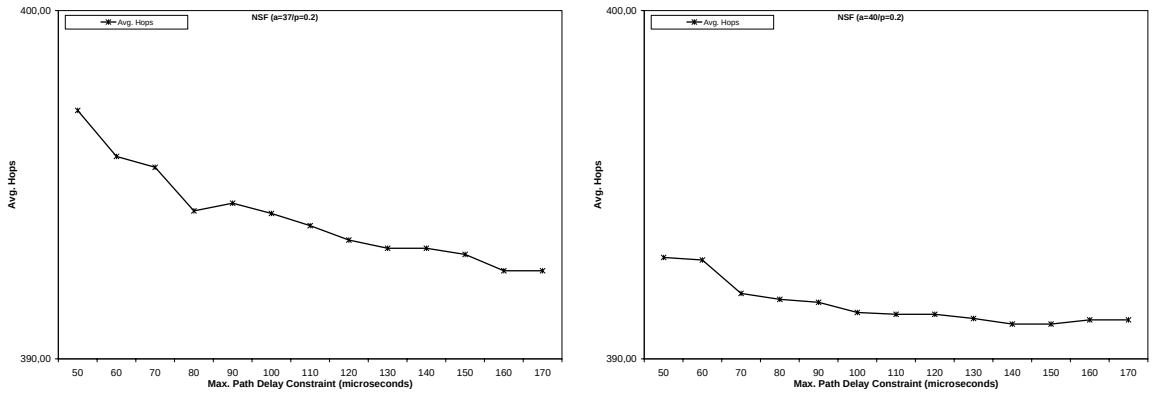
Figures 6.6 and 6.7 show the evolution of the end-to-end path delay with the constraint imposed on the maximum allowable end-to-end path delays for both topologies. As it has been highlighted in section 6.2, when setting the end-to-end path delay constraint to

50 $\mu seconds$, the observed *absolute* maximum end-to-end path delay (i.e. the maximum observed among all the paths in the layouts obtained for the set of 25 traffic matrixes for a given load level and θ_q) is far lower than the value imposed by the constraint. This is due to the way those constraints were simplified in order to be able to calculate flow shifts for the MFD algorithm in a link-by-link basis taking into account the path flows traversing each link. A factor to analyze is then how the absolute maximum path delay for any path in the layout relates to the imposed path delay constraint.



(a) VTHD, $a = 115$ (b) VTHD, $a = 120$

Figure 6.8: Average Quantity of Hops for VTHD Network Topologies Using MFD Algorithm, as a Function of Delay Constraint

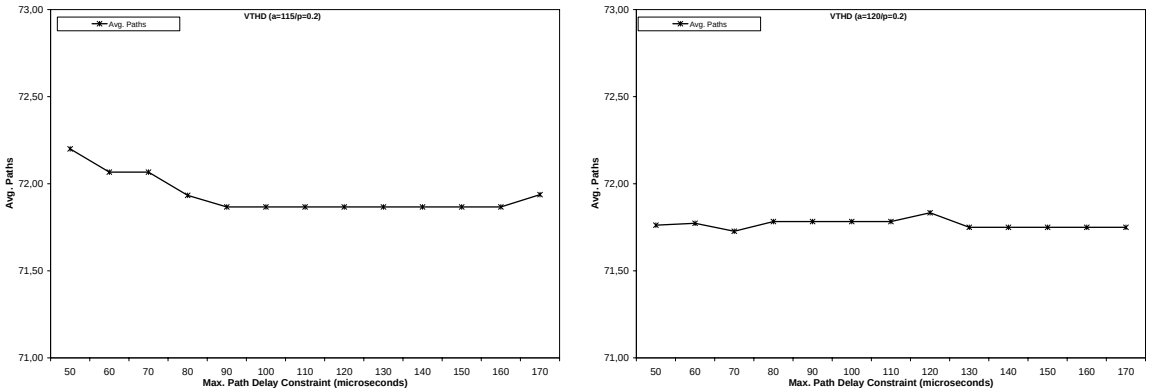


(a) NSF, $a = 37$ (b) NSF, $a = 40$

Figure 6.9: Average Quantity of Hops for NSF Network Topologies Using MFD Algorithm, as a Function of Delay Constraint

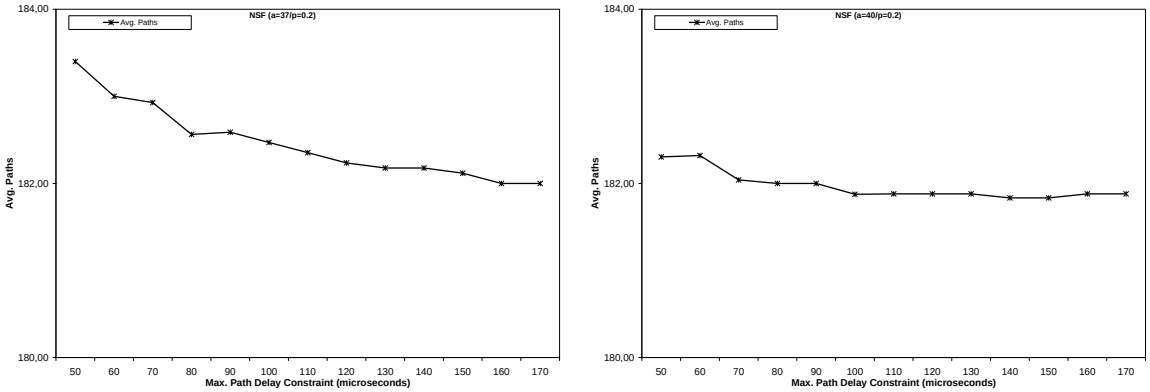
As we would expect, the maximum path delay ($\max_{k,q}\left\{\tau_q^k\right\}$) obtained in any path of the calculated layout increases as the path delay constraint (θ_q) are made less tight. We see that, even when increasing θ_q to 160 $\mu seconds$ for a high load situation, $\max_{k,q}\left\{\tau_q^k\right\}$ stays

below the originally set constraint of 50 $\mu seconds$. For a medium load situation ($a = 120$ for VTHD and $a = 40$ for NSF), the $\max_{k,q}\left\{\tau_q^k\right\}$ reach later the limit imposed by the originally set constrain. We can also observe that for NSF network, the $\max_{k,q}\left\{\tau_q^k\right\}$ crosses the originally set constraint of 50 $\mu seconds$ for lower values of the path delay constraint than it does for VTHD network.



(a) VTHD, $a = 115$ (b) VTHD, $a = 120$

Figure 6.10: Average Quantity of Paths for VTHD Network Topologies Using MFD Algorithm, as a Function of Delay Constraint

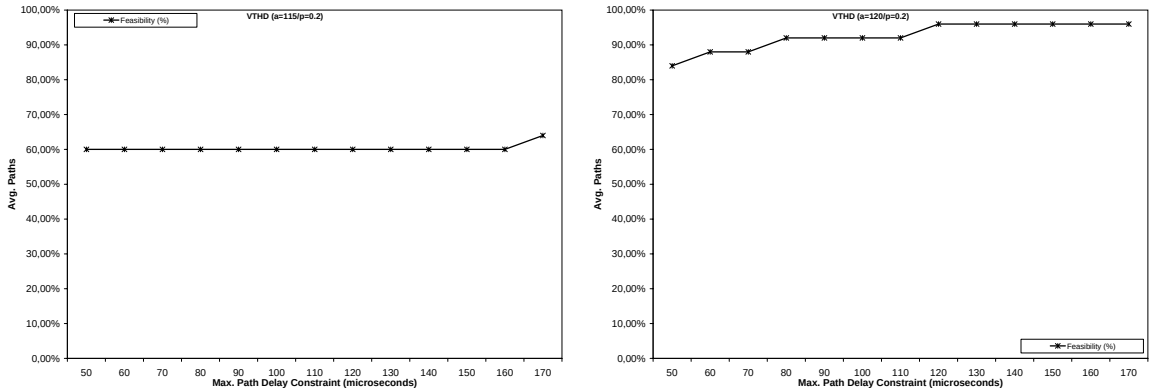


(a) NSF, $a = 37$ (b) NSF, $a = 40$

Figure 6.11: Average Quantity of Paths for NSF Network Topologies Using MFD Algorithm, as a Function of Delay Constraint

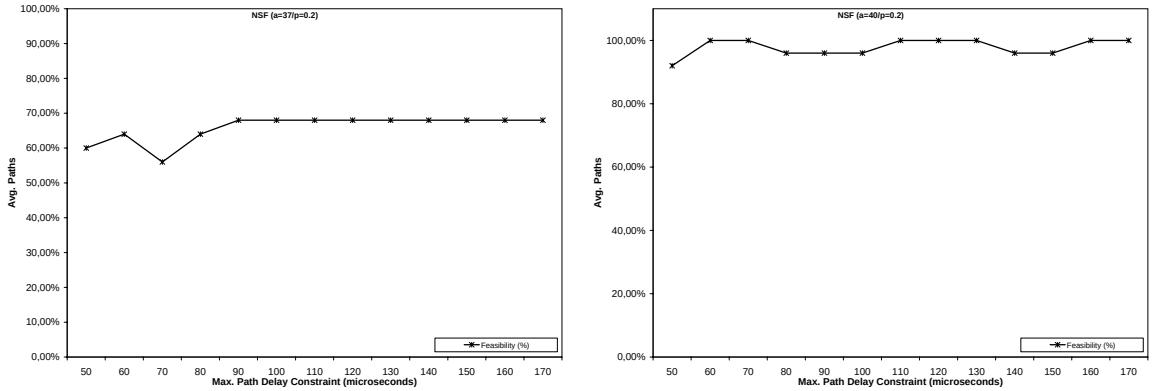
This can be explained by the fact that the average path length for NSF network is larger than the average path length for VTHD network. The fact that the obtained path delays are always lower than the imposed constraints allows us to set less tight constraints, allowing the concentration of traffic in less paths, and consequently lowering the number of required hops (and indirectly of paths), as it can be seen in Figures 6.8, 6.10, 6.9 and 6.11. This results

in an further improvement on the quality of the results, without violating the originally set constraints. Further study to find a relationship between the values obtained for the path delays for a given set of path delay constraints may help to set engineering rules for the design of MPLS layouts. This relationship depends on the topology so an explicit relationship would be difficult to establish, although we see from the figures that a linear function yields a fairly good approximation.



(a) VTHD, $a = 115$ (b) VTHD, $a = 120$

Figure 6.12: Feasibility Rate for VTHD Network Topologies Using MFD Algorithm, as a Function of Delay Constraint



(a) NSF, $a = 37$ (b) NSF, $a = 40$

Figure 6.13: Feasibility Rate for NSF Network Topologies Using MFD Algorithm, as a Function of Delay Constraint

Finally, looking at the feasibility ratios in Figure 6.12 and 6.13, as we would expect, it can be seen that they improve when the path delay constraints are made less tight. The reason why this happens is simple: as the path delay constraints are relaxed, the allowed *reduced* capacity on each link increases, also increasing the total available capacity in the network,

so that traffic matrixes that were infeasible before now become feasible.

6.4. Conclusions

In previous Chapter we developed two heuristic methods to solve the MPSFAP problem. We also tested implementations of both heuristics on small network topologies in order to evaluate the quality of the results obtained, given the problem size limitations imposed by the deterministic solvers. From results obtained in Chapter 5 we conclude that the TS and MFD algorithms implementing generic and ad-hoc heuristics respectively yield approximate results within an acceptable tolerance from the optimal solutions. In the present Chapter, we explore the behavior of both algorithms and the cost functions proposed on large network topologies. Results are obtained for two load levels, corresponding to heavy and mild load in the network.

Regarding the performance of the algorithms proposed in previous Chapter, for the same topologies and under the same load conditions, we observe that the MFD algorithm performs better than the TS algorithm for both aspects, quality of the solution and execution time. With respect to the quality of the solutions obtained my MFD, it can be observed that the value of the MPSFAP1 cost function is lower in all cases, meaning that a lower number of hops and paths are used in average. On the other hand, we observe that MFD has an infeasibility ratio which is higher than the one observed for TS. This is explained by the fact that the simplification operated on the end-to-end path delay constraint when implementing the MFD algorithm imposes more restrictive limitations to the usable capacity on the individual links than the TS algorithm. This can be observed when comparing the maximum end-to-end path delay obtained from both algorithms for each one of the 25 traffic matrixes in the test set for any load level. While the maximum path delay values obtained with TS are almost always close to the imposed constraint, the values for the MFD algorithm are always far from such limit. This drives us to further explore solutions with the MFD algorithm for less restrictive path delay constraints, which we do in the second part of this Chapter. Further, with respect to execution times, we can observe that MFD algorithm always find better quality solutions than TS in dramatically lower times than TS, due to the inexpensive nature of the operations performed over only a few paths on each iteration of the MFD. The execution times observed with the MFD algorithm are compatible with an on-line operation of a close control loop, allowing us to reduce the time required to calculate optimal layouts and to dimension the network at shorter timescales. However, it is not practical to recalculate the optimal layout in a large network too many times too often, since TE mechanisms at shortest timescales applied on local scopes can deal appropriately with the dynamic adaptation needed for the observed variations at those timescales, instead of resorting to global changes which are not justified in terms of costs

for the operator. The quality of the observed results, together with the execution times are key factors, which determined us to further develop the MFD algorithm in future research directions, leaving the TS algorithm behind. In particular, a version of the MFD algorithm is adapted to solve the MRPSFAP problem in Chapter 8.

Regarding the behavior of the proposed MPSFAP cost function, the study of results obtained by using the MFD algorithm on large network topologies show that when imposing less restrictive constraints under any load condition, the average quantity of hops (and indirectly the average quantity of paths) gets further reduced. As expected, reducing the load conditions on a given network topology, also reduces the number of required hops and paths for a given traffic matrix. It is important to note that in all cases the path multiplicity required to connect any node pair when calculating the layouts with MPSFAP1 objective is close to 1, showing the interest of the proposed cost function as representative of the cost objectives for layout optimization from an operational point of view.

7. Contribution to Reconfiguring MPLS Networks: Design of Reduced Reconfiguration and Complexity Layouts

7.1. Motivations and Previous Work

In Chapter 4 the problem of dimensioning an MPLS layout has been addressed. Given a traffic demand and a physical network topology, the objective is to find the set of *source routed* paths and the corresponding flow allocation that minimizes the cost of operation. We have considered so far that the traffic demand is known (e.g. by means of measurement and inference from those measures) and that we want to obtain a unique optimal layout for that traffic matrix. In this context, we consider that the cost of operation is related to the layout complexity in terms of quantity of hops and paths that have to be established and maintained by the operating system. The dimensioning problem is then formulated as a mixed-integer non-linear constrained multicommodity flow allocation problem (MPSFAP) including cost functions taking into account the above mentioned cost issues [16].

On long timescales, traffic demands are observed to present systematic long-term traffic variations on a daily basis [68, 69]. The design of an optimal layout for a given traffic matrix constitutes then a point of operation during the periods where the traffic is identified to be *static* with respect to the considered timescale. Small traffic variations during this cycle would be accommodated by TE mechanisms taking action at shorter timescales. However, when the traffic matrix presents significant changes (as those produced when passing from the current static period to the next one), the currently operational layout is not able to handle the new traffic conditions efficiently anymore and has to be recalculated. In the present chapter we consider that there is an additional cost for the network operation and maintenance in *reconfiguring* the network to the new layout. Indeed, depending on the routing policy being implemented, whether additional resources have to be made available or service disruption times have to be accepted in order to manage the transition from the current layout to the next one. In this context, the layout has to be recalculated to be optimal with respect to the layout complexity (in the sense of MPSFAP problems) *and* with respect to the *reconfiguration* complexity.

Reconfiguration studies have been carried out mainly in the context of WDM layouts [25, 70, 71] and joint MPLS and WDM layout optimization [72], as well as in the field of general optimal routing [73]. Objective functions considered in off-line approaches minimize the

number of lightpath additions and deletions [25] or a trade-off among the reconfiguration and the other objectives in the layout design, without considering QoS constraints. In [25] a new virtual topology is obtained by choosing the layout which minimizes the total number of added or deleted lightpaths among all the possible layouts. An optimal layout is calculated for the old and new demand matrixes. The value of the objective function for the new layout is then included in a new formulation of the problem for the new layout as a constraint. On-line approaches [70, 71] consider the objective of minimizing the lightpath additions and deletions in order to adapt the virtual layout to traffic changes. In [70], low and high watermarks are associated to each lightpath. The load on every lightpath is observed, and only one operation (i.e. adding or deleting a lightpath) is allowed at the end of the adaptation cycle. In [71], trade-offs between the reconfiguration costs and the other objectives are proposed when calculating a path for an arriving demand. In a different approach that can be assimilated to the reconfiguration problem, [72] and [73] try to find optimal layouts for MPLS over WDM networks and for the general IP routing, for a given objective and for a set of traffic demands lying within a *polytope*. This was first addressed in [89] and [90] and is commonly referred as *Muti-time Period Network Design*. Instead of calculating optimal layouts considering reconfiguration objectives, a unique optimal layout is calculated with respect to some performance objective, and for a set of demand matrixes inscribed in a polytope representing all the cycles of traffic variation. In this context, the optimal layout calculated is expected to be a good point of operation for the set of traffic matrixes in the polytope, without needing to be recalculated and in consequence rebuilt or modified. However, there is a trade-off between the extension of the polytope for which the layout stays optimal and the optimality of the layout with respect to the considered objective for any particular matrix inscribed in it. In order to have a wide range of valid demands considered into a single optimal layout, the calculated layout will be suboptimal with respect to the individual optimal layouts that would be obtained for the individual matrixes in the set, with the same objective. The suboptimal layout will then be *farther* from the individual optimal layouts as the size of the considered polytope increases. This leads to a situation similar to network overdimensioning. In particular, if the objective is to obtain an optimal layout in the sense of MPSFAP problems, then the number of hops and paths would have to be increased in order to consider a larger number of possible matrixes. In [45] and [75] the metrics of traditional routing protocols in a pure IP environment are modified in order to adapt the network to varying traffic conditions. However, in the context of this thesis, Interior Gateway Protocols (IGPs) are not considered as appropriate tools to allow evolved TE mechanisms to be implemented on NG IP Networks.

In a different but complementary field of research that has recently started to attract attention, progressive techniques to establish the newly calculated layout are being studied. Once the new layout calculated to be optimal with respect to the complexity of the layout and the complexity of the reconfiguration, we still need a way to set up the transition in a

progressive non-disruptive and least cost way. In [74] the new layout is considered to have been already calculated, and a way of minimizing the transition costs is formulated.

All the cited work lead to more or less complicated MILP, MINLP and LP problem formulations. The cost of reconfiguration in WDM networks is related to wavelength commutation, so lightpath deletions are as costly as lightpath additions. Besides, we consider that including reconfiguration objectives in the layout calculation is a good approach to a more efficient network adaptation to varying traffic conditions.

However, in the context of source-routed IP paths (e.g. MPLS paths) only path additions are costly, since depending on the routing strategy spare resources have to be made available or service disruption times tolerated in increasing proportions of the quantity of paths to *replace*. The cost of reconfiguration is then considered to be related to the number of paths (with its associated weight) that have to be *added* when building up the new layout. The number of paths to be deleted represent only a marginal cost and can be neglected (i.e. deleting paths reduces the total cost of the layout). The reconfiguration problem is formulated as the dynamic version of the MPSFAP problems defined in Chapter 4. The Minimum Reconfiguration, Path Set and Flow Allocation Problem (MRPSFAP) is then defined as a Mixed-Integer Non-Linear Program (MINLP) whose objective is to minimize a function of the difference between the current and the calculated layout and also a function of the layout complexity as defined in MPSFAP, under end-to-end QoS constraints for the new layout. As before, two cost functions are proposed correspondingly to MPSFAP1 and MPSFAP2. A unique class of service is considered first, and QoS guarantees are given for it. The model is then extended to consider multiple classes of service.

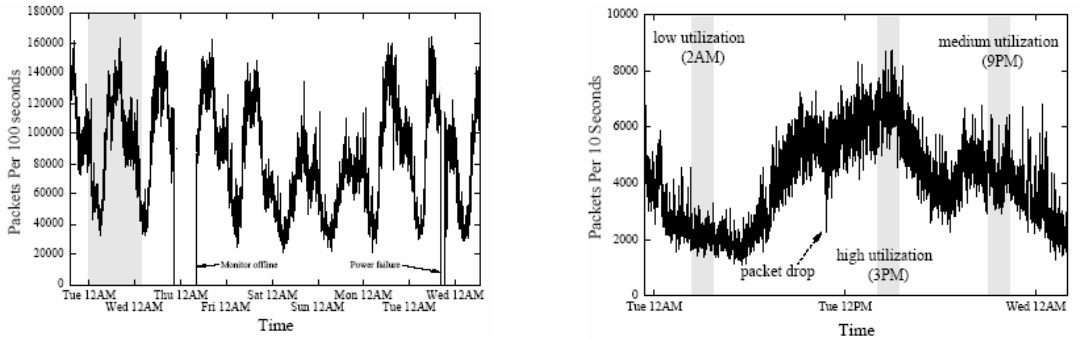
7.2. Network Model Notation: Extensions for Dynamic Traffic Conditions

In this section we recall the definitions from section 4.2 and redefine them as necessary to be used in the context of the reconfiguration problem. The network is again represented as a graph, and the arc-incidence matrix $\mathbf{A}$ represents the set of all possible paths on the physical topology as before. $\mathbf{W}$ represents the K-dimensional single column matrix of weights associated to each path. Both matrixes don't change with time (or *instance*) considered, as they represent physical characteristics of the underlying network. In what follows, we will denote with t the current instant and $(t+1)$ the instant for which we want to calculate the new optimal layout.

Demands: Characterizing the Traffic Dynamics

Traffic measured at packet level [68] (Figure 7.1) and at flow level [69] (Figure 7.2) show systematic long-term cyclic variations during the day and between weekdays and weekends. In particular at the flow level, we can then identify periods that can be each represented by

a different traffic matrix. Assuming that the current layout was calculated for the traffic matrix at the instant t, $\mathbf{D_t}$, the new layout will be calculated for the traffic matrix at the instant $(t+1)$, $\mathbf{D_{(t+1)}}$.



(a) Traffic per 100 seconds for 9 days packet trace (b) Traffic per 10 seconds for 27 hours packet trace

Figure 7.1: Traffic measures at packet level for different timescales [68]

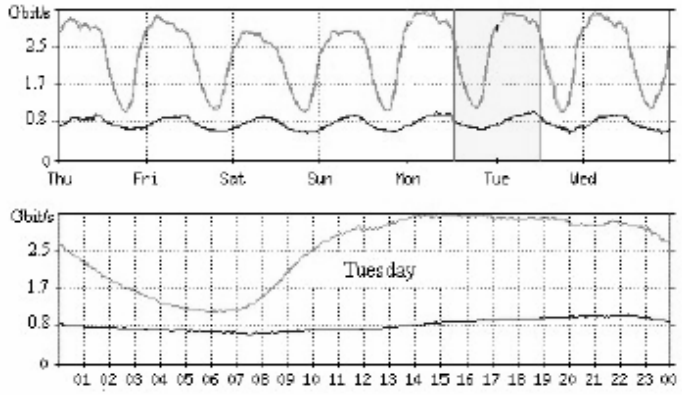


Figure 7.2: Weekly and Daily Traffic Profile on a OC192 Link [69]

Correspondingly, the single column matrix representing the path flows for the new layout will be now denoted as $\mathbf{B_t}$ for the current layout and $\mathbf{B_{(t+1)}}$ for the new layout. The relationships (4.4) and (4.5) get modified as follows:

$$\mathbf{R} \cdot \mathbf{B}_{(\mathbf{t+1})} = \mathbf{D}_{(\mathbf{t+1})} \tag{7.1}$$

$$\mathbf{X}_{(\mathbf{t+1})} = \mathbf{A} \cdot \mathbf{B}_{(\mathbf{t+1})} \tag{7.2}$$

respectively for the instant $(t+1)$ with $\mathbf{R}$ defined as before.

7.3. Building the Cost Functions

Considering traffic dynamics, the cost of operation in large networks is related not only to the complexity of the operating layout as we stated in Chapter 4, but also to the reconfiguration complexity. In fact, establishing a layout from scratch has OAM costs related to the number of paths and the length of those paths (e.g. in hops) to be set up and maintained by the management system. In a dynamic environment however, the network is not likely to be set up from scratch: a currently operational layout is in place, and there are OAM costs associated to any change that has to be made to that layout. In current operational environments, the layouts are recalculated independently for each traffic matrix and then rebuilt through one of two strategies [10]: *make-before-break* and *break-before-make*. Make-before-break strategy establishes all the paths in the new layout and starts routing traffic through them before start deleting the paths in the old layout. The advantage is that no service disruption is produced with such strategy, but enough spare resources have to be made available in order to be able to route all the traffic during the transition; at the same time, the management system has to deal with roughly twice the number of paths until the new layout is set up. On the other hand, a break-before-make strategy deletes all the paths in the layout and rebuilds it as it if were from scratch. The advantage of this approach is that no spare resources are needed during the transition, but service disruption is unavoidable; at the same time, the management system has to deal with a whole layout deleting and set-up. The larger the network, the more spare resources have to be made available with the former strategy or larger service disruption times will be expected with the later strategy. In all cases, the management system deals with a large number of paths whenever a transition is made. Our approach considers the cost of reconfiguration as an objective when calculating the new layout, as well as the already stated objective of layout complexity. No matter what strategy is used then to set up the transition, its cost for the OAM are minimized. The order in which the paths are established or deleted during the transition constitutes a complementary research direction.

Assuming that the optimal layout for the traffic demand $\mathbf{D}_{(\mathbf{t})}$ at the instant t has been found (i.e. whether by solving a MPSFAP1 problem if $t = 0$ or a MRPSFAP problem if $t \neq 0$), $\mathbf{B}_{(\mathbf{t})}$ and $\mathbf{H}_{(\mathbf{t})}$ represent the path flow distribution and the set of used paths in the

optimal layout. Recalling from (4.6), the objective will be then to minimize a function of the general form:

$$\text{Minimize} \left[\varphi \sum_{q=1}^{Q} \sum_{k=1}^{K_q} w_q^k h_{q,(t+1)}^k + \nu \sum_{q=1}^{Q} \sum_{k=1}^{K_q} w_q^k s_{q,(t+1)}^k \right] \tag{7.3}$$

where h_k^q is a binary variable defined as in (4.7) for the new layout (i.e. instant $(t+1)$), and the weights associated to each path are calculated as in Section 4.3. The first term in (7.3) represents the objective of reduced complexity for the new layout as in the MPSFAP problems. The second term represents the objective of reduced reconfiguration complexity, and is a function of the current layout (instant t) and the layout being calculated (instant $(t+1)$). φ and ν are factors to control the importance of each individual objective. We now need a way to define the matrix $\mathbf{S_{(t+1)}}$ as a measure of the difference between the current and new layouts, in order to measure what we consider as the reconfiguration cost.

If the dispersion in the end-to-end path delays is an issue when calculating the new layout, a third term can be added to the general objective in (7.3) as we proposed in (4.8). The effects of such a term in the objective function were studied in Chapter 4 and will not be addressed here again. In what follows, we will only define the MRPSFAP problem to consider objective functions of the form (7.3), as the results obtained for MPSFAP2 problem can be extended to the new layouts calculated by including this third term. We focus then on the reconfiguration aspects of the new layout, given the current operational one.

Minimizing Layout Reconfiguration: Differential Index

As we stated before, the cost of reconfiguration in the context of MPLS layouts is related to the quantity of paths (and their lengths in hops) that are to be established in the new layout with respect to the current operational layout. The addition of a path, even when replacing an existent one imposes a cost to the management system and should be avoided. On the other hand, we consider that the deletion of a path imposes only a marginal cost to the OAM, and should be encouraged whenever the deletion is not being replaced by a new path. The entries in $\mathbf{S_{(t+1)}}$ or *differential indexes* will be then binary variables indicating wether the path a_q^k is an addition (i.e. receives flow) in the layout being calculated or it is an existent one (i.e. keeps some positive flow) or it is being deleted (i.e. looses its flow):

$$s_{q,(t+1)}^k = \begin{cases} 1 & \text{if } h_{q,(t+1)}^k = 1 \wedge h_{q,(t)}^k = 0 \\ 0 & \text{otherwise} \end{cases} \tag{7.4}$$

for $k = 1, \ldots, K_q$ and $q = 1, \ldots, Q$. In other words, $s_{q,(t+1)}^k$ can be defined as a function of $h_{q,(t)}^k$ and $h_{q,(t+1)}^k$:

$$s^k_{q,(t+1)} = h^k_{q,(t+1)}(h_{q,(t+1)} - h_{q,(t)}) \tag{7.5}$$

where $h_{q,(t)}$ are the binary variables indicating if a given path a^k_q was used in the previous layout, and is an input of the new problem.

7.4. Setting QoS Guarantees

The constraints introduced for the MPSFAP problems in section 4.4 are kept. In particular, the end-to-end path delay (4.11) and throughput (4.10) guarantees must be ensured for the new layout.

7.5. Formulation: Minimum Reconfiguration, Path Set and Flow Allocation Problem (MRPSFAP)

Given the network physical topology $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the currently operational layout $\mathbf{B_{(t)}}$ and $\mathbf{H_{(t)}}$ (optimal for some demand matrix $\mathbf{D_{(t)}}$) and the new demand matrix $\mathbf{D_{(t+1)}}$, the problem is to find the new set of paths $H_{(t+1)}$ and the flow allocation over those paths $\mathbf{B_{(t+1)}}$ which minimize a cost objective, complying to some QoS constraints. We formulate the problem of designing the optimal layout w.r.t. the complexity of the new layout and the complexity of reconfiguration from the current layout, while meeting QoS constraints (i.e. and end-to-end path delay constraints and available bandwidth per node pair). The *Network Design Multicommodity Flow Allocation Problem* in its matrix form is stated as the MRPSFAP1 problem below.

The objective function (7.6) includes the term considering the new layout complexity ($(t+1)$), and the term considering reconfiguration complexity for the transition from the old to the new layout ($t \rightarrow (t+1)$). We consider that calculating the path weights as the number of hops each path uses is a good choice to take care of OAM costs for both, layout complexity and reconfiguration complexity. In fact, the management system (whether through manual setup or by means of a signaling protocol) effort to set up each path is proportional to the number of links a path is traversing. From the performance point of view, it is also preferable to choose shorter paths (in terms of hops) instead of longer paths. Constraints (7.7), (7.8), (7.9) and (7.10) are equivalent to the constraints included in MPSFAP problems, applied on the new layout being calculated. The relation (7.5) tying $\mathbf{S_{(t+1)}}$ to $\mathbf{H_{(t)}}$ and $\mathbf{H_{(t+1)}}$ is expressed as a constraint (7.11). The reason is that both $h^k_{q,(t+1)}$ and $s^k_{q,(t+1)}$ are variables that the solver will choose to minimize the cost function. As both are binary variables, through the inequality we ensure that the equality will be true in the optimum. In section 7.6 we compare results obtained from MRPSFAP and MPSFAP1 for NET2 topology.

MRPSFAP 1
Minimum Reconfiguration, Path Set and Flow Allocation Problem

Given:

$$\mathbf{A}, \mathbf{C}, \mathbf{D_{(t+1)}}, \mathbf{H_{(t)}}, \mathbf{\Theta}, \mathbf{\Delta}, \mathbf{W}$$

minimize :

$$\varphi^t \mathbf{W} \cdot \mathbf{H_{(t+1)}} + \nu^t \mathbf{W} \cdot \mathbf{S_{(t+1)}} \tag{7.6}$$

subject to:

$$\mathbf{X_{(t+1)}} = \mathbf{A} \cdot \mathbf{B_{(t+1)}} \leq \mathbf{C} \tag{7.7}$$

$$\mathbf{R} \cdot \mathbf{B_{(t+1)}} = \mathbf{D_{(t+1)}} \tag{7.8}$$

$$h^k_{q,(t+1)} \sum_{i=1}^{M} \frac{\lambda a^k_{q,i}}{C_i - x_{i(t+1)}} \leq \theta_q \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.9}$$

$$b^k_{q,(t+1)} \leq h^k_{q,(t+1)} \delta_q \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.10}$$

$$(h^k_{q,(t+1)} - h^k_{q,(t)}) h^k_{q,(t+1)} \leq s^k_{q,(t+1)} \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.11}$$

$$h^k_{q,(t+1)} \in \{0, 1\} \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.12}$$

$$s^k_{q,(t+1)} \in \{0, 1\} \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.13}$$

$$b^k_{q,(t+1)} \geq 0 \qquad k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.14}$$

7.6. Preliminary Results on MRPSFAP and Model Validation

The MRPSFAP problem, as the MPSFAP problems, is a *network design multicommodity flow problem*, with the binary variables $h^k_{q,(t+1)}$ and $s^k_{q,(t+1)}$ deciding on wether a paths is going to be used or not in the new layout, based on usage of those paths in the previous layout. The resulting MRPSFAP is a MINLP problem, and as before, the non-linearities are given by the end-to-end path delay constraints imposed on the new layout. The MRPSFAP problem presents a larger number of variables: $3K$ instead of $2K$ in MPSFAP problems, and $3K + M + Q$ constraints, instead of the $2K + M + Q$ constraints present in the MPSFAP problems. As the MPSFAP problems, the MRPSFAP problem is $\mathcal{NP}$-*complete*, what reduces the size of the network topologies that can be solved through deterministic algorithms.

In the present chapter, due to the size limitations imposed by the nature of the problem, we evaluate some preliminary results limited to the NET2 topology (NET1 doesn't offer a significant difference for reconfiguration) in order to obtain some insight in the behavior of the objective function and validate the correctness of the proposed cost model. In Chapter 8 we resort to an heuristic method derived from the Modified Flow Deviation Algorithm

(MFD) presented in Chapter 5.

7.6.1. Traffic Matrixes: Considering the Traffic Dynamics

Traffic matrixes were generated to evaluate the MPSFAP problems in section 4.6.1, following a method inspired in [25]. To model the dynamics of the demands over a set of identified *static* periods, we make evolve each of the generated matrixes by operating a change in the demand value for a given proportion p_v of the node pairs. The new value of the demand for a changed node pair is uniformly distributed within the interval $[0, \frac{C_q}{a}]$ or the interval $[0, \frac{C_q Y}{a}]$, depending on the interval in which that node pair had its demand distributed when generating the original traffic matrix. We generate then a series of 24 demand matrixes for each original randomly generated matrix, each new matrix evolving from the previous one in the series. Figure 7.3 shows the process of generating a series of matrix evolutions from an original matrix.

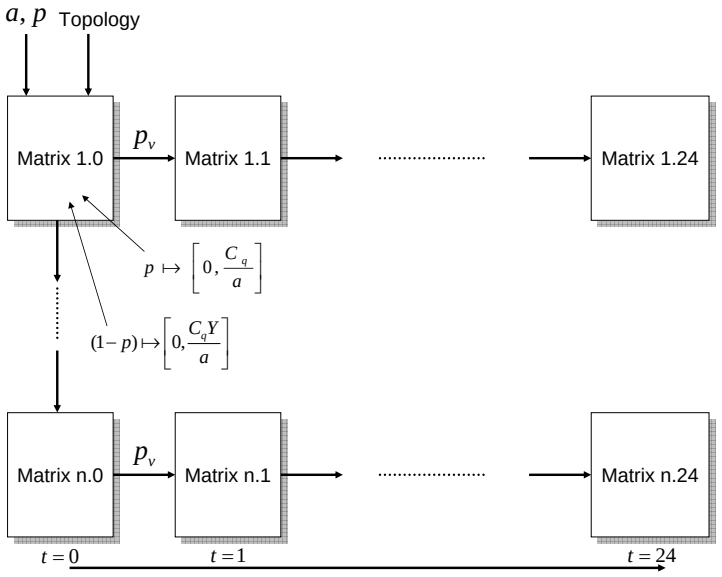


Figure 7.3: Overall process to generate the test set of demand matrixes (static and dynamic)

7.6.2. Results and Analysis

The MRPSFAP problem has been modelled into AMPL format in order to be submitted to the MINLP solver available at the NEOS server on the Internet. The model in AMPL language for the MRPSFAP problem can be found in Appendix B. The procedure we follow is depicted in Figure 7.4. For a given traffic matrix at the instant $t = 0$ denoted by $M_{1.0}$

and randomly generated as described in section 4.6.1, we produce an optimal layout in the sense of MPSFAP1 problem. We denote this optimal layout by S_0. Then, we solve the first matrix evolution $M_{1.1}$ for the instant $(t+1)$ using MPSFAP1 to obtain a new layout without correlation to the current layout S_0. We denote this optimal layout obtained with MPSFAP1 by S'_1. At the same time, we solve the same matrix $M_{1.1}$ using MRPSFAP to obtain a new optimal layout correlated to the currently operational. We denote this optimal layout by S_1. The process is repeated until all the evolutions in the series for the considered matrix have been solved. The obtained results are then compared regarding the total number of hops and paths accumulatively added through the process for each series of traffic matrix changes, as well as with respect to the total number of hops and paths used by each one of the optimization criteria.

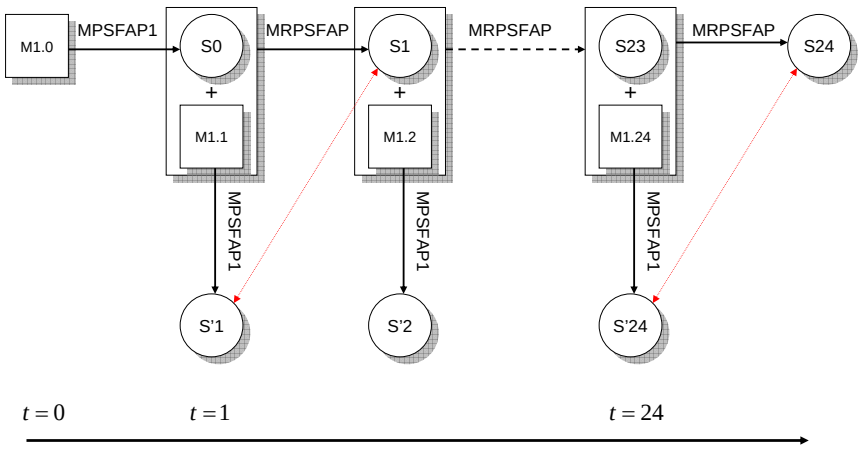
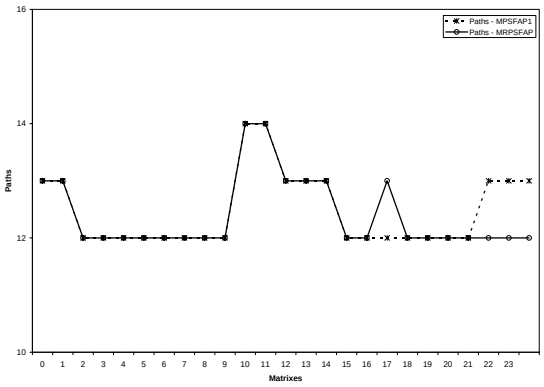


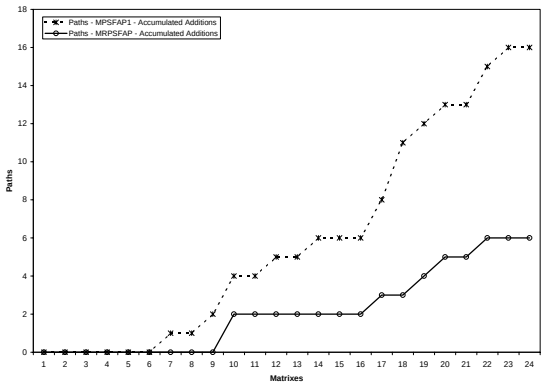
Figure 7.4: Overall process to compare layouts obtained from MPSFAP1 and MRPSFAP for a series from an original matrix

Due to the number of demand matrixes in each series, we limit our study to the NET2 topology shown in Figure 4.3 and described in section 4.6.2. To generate the traffic matrixes, the load level parameter is set to $a = 5$ (which corresponds to a high load level), the proportion of nodes being allowed a higher load is set to $p = 0.2$, and the number of nodes pairs having their load changed from matrix to matrix is set to $p_v = 0.4$ (meaning that 40% of the node pairs are having their demand values changed from the matrix at t to the matrix at $(t+1)$) for the M_1 series, and $p_v = 0.2$ for the M_2 series. We consider that results on NET2 are representative of results that could be obtained on a similar test topology like NET1, as

it shows a higher connectivity (density of 2.5) and consequently a higher number of paths that can be changed. The importance of the complexity and reconfiguration objectives are equally weighted ($\varphi = \nu = 1$). The end-to-end path delay constraints are in all cases kept as in Chapter 4. Next, results obtained for two traffic matrixes and their corresponding evolutions are shown.

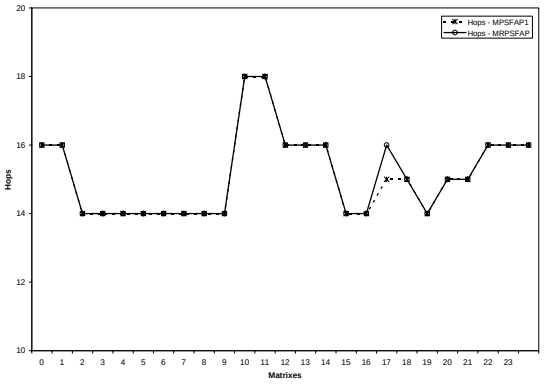


(a) Paths, $a = 5, p = 0.2, p_v = 0.4$

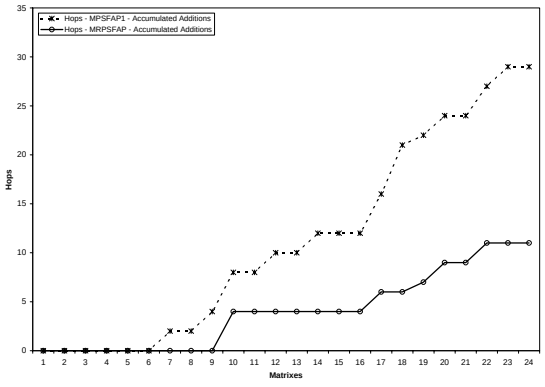


(b) Accumulated path additions, $a = 5, p = 0.2, p_v = 0.4$

Figure 7.5: Used paths and accumulated path additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1



(a) Hops, $a = 5, p = 0.2, p_v = 0.4$

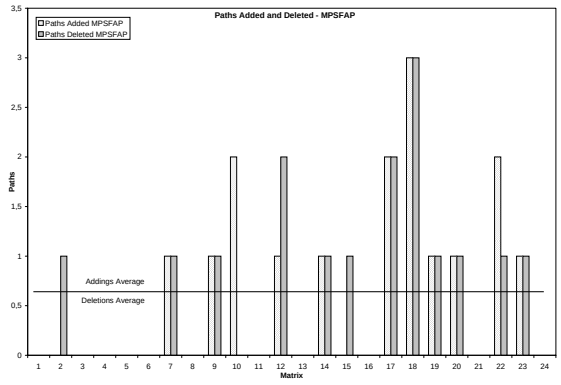


(b) Accumulated hops additions, $a = 5, p = 0.2, p_v = 0.4$

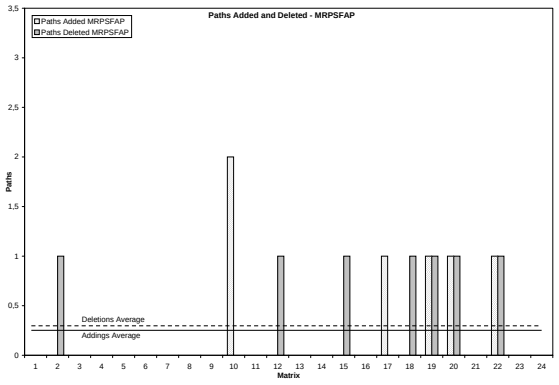
Figure 7.6: Used hops and accumulated hop additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1

In Figures 7.5(b) and 7.6(b), we observe that the number of paths and hops *accumulatively added* through the optimization of the reconfiguration complexity operated by MRPSFAP for the 24 traffic matrixes in the M_1 series is significatively lower than the number of addi-

tions operated by independently optimizing only the layout complexity through MPSFAP1.

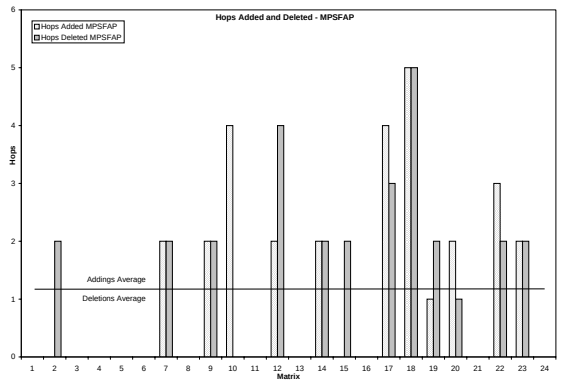


(a) MPSFAP1, $a = 5, p = 0.2, p_v = 0.4$

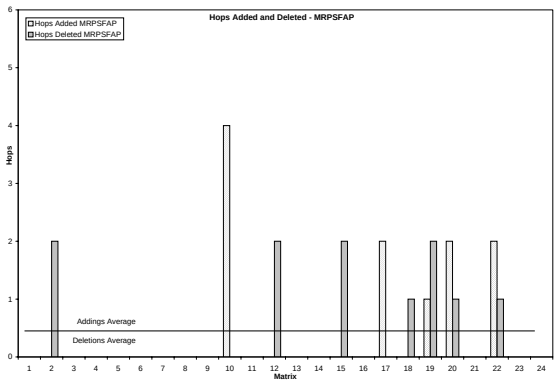


(b) MRPSFAP, $a = 5, p = 0.2, p_v = 0.4$

Figure 7.7: Path additions and deletions for MPSPAF1 and MRPSFAP for demand matrix series M_1



(a) MPSFAP1, $a = 5, p = 0.2, p_v = 0.4$

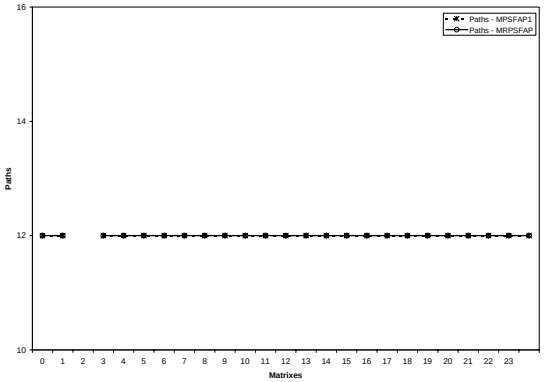


(b) MRPSFAP, $a = 5, p = 0.2, p_v = 0.4$

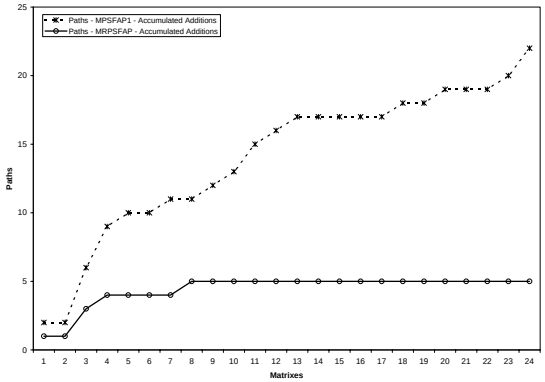
Figure 7.8: Hop additions and deletions for MPSPAF1 and MRPSFAP for demand matrix series M_1

By examining Figures 7.5(a) and 7.6(a) we see that layouts obtained by both MPSFAP1 and MRPSFAP use almost the same quantity of hops and paths through the whole series. The last shows that MRPSFAP optimizes the layout complexity, as well as the reconfiguration complexity during transitions, which is the objective we aim at for the design of MPLS layouts under dynamic traffic conditions. Also, as the traffic conditions vary significatively from one matrix to the next in M_1 (since $p_v = 0.4$ in this series), the number of hops and paths also changes to adapt to those variations. However, on some cases MRPSFAP appears to use more hops and paths than MPSFAP1, but we see that those cases correspond

to the transitions when additions have to be done and in that case, MRPSFAP is choosing to use one more path (or a longer one in hops) to keep the number of additions low. It is important to note that from the results obtained, we verify that the fact of including only the weighted path additions in the cost function helps the network to adapt to a decrease in the load level by allowing it to reduce the number of paths in the optimal layout. If path deletions were penalized as the additions, MRPSFAP would rather choose a higher number of paths than required for a given traffic load, leading to a non-efficient resource usage.

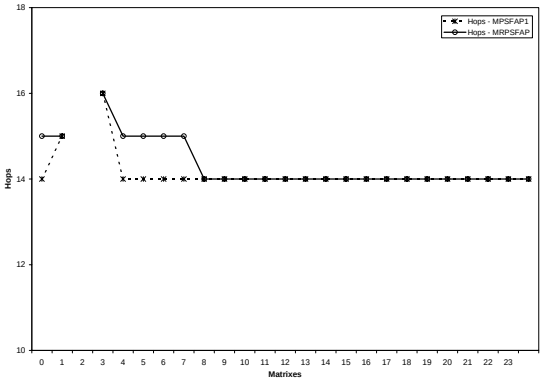


(a) Paths, $a = 5, p = 0.2, p_v = 0.2$

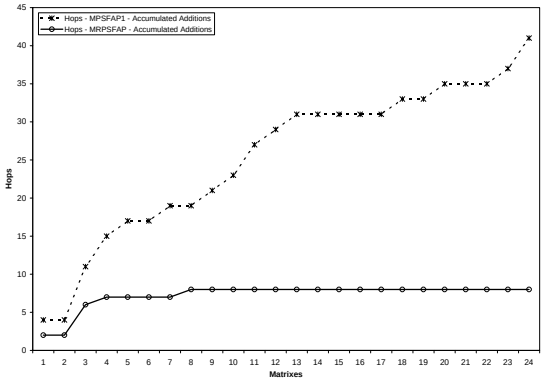


(b) Accumulated path additions, $a = 5, p = 0.2, p_v = 0.2$

Figure 7.9: Used paths and accumulated path additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_2



(a) Hops, $a = 5, p = 0.2, p_v = 0.2$

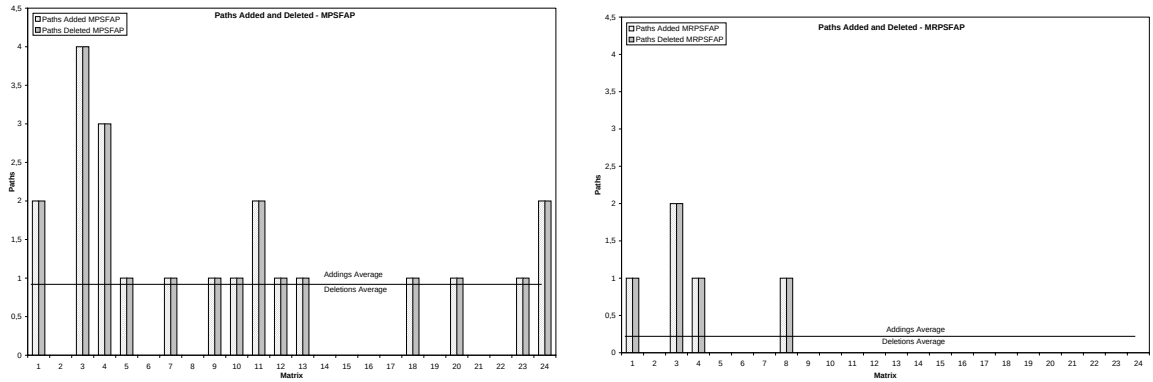


(b) Accumulated hops additions, $a = 5, p = 0.2, p_v = 0.2$

Figure 7.10: Used hops and accumulated hop additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_2

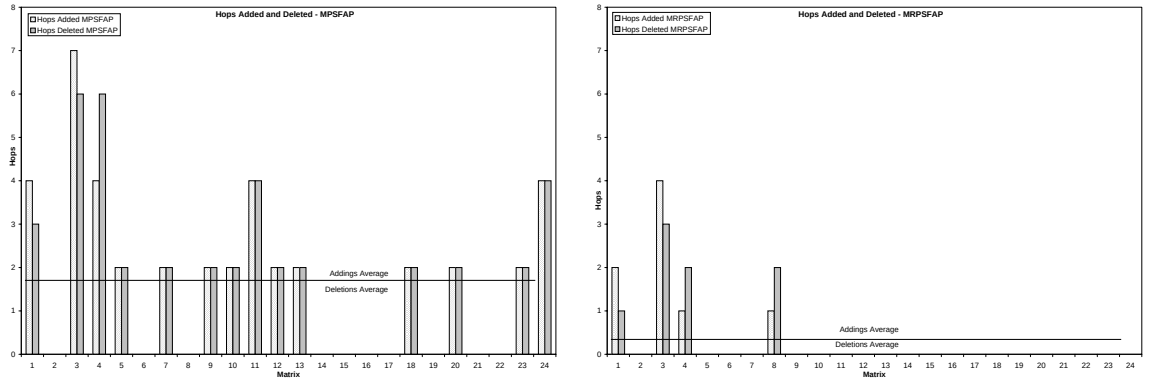
Figures 7.7(a), 7.7(b), 7.8(a) and 7.8(b) show the detailed path and hops additions and

deletions operated by both MPSFAP1 and MRPSFAP for the 24 matrixes in the M_1 series. We see that the addition and deletion activity is always lower for MRPSFAP than for MPSFAP1 for both paths and hops. In conclusion, MRPSFAP requires in general less changes (i.e. additions) than MPSFAP1 to realize a series of traffic adaptations, and when those changes have to be done, they are in general *shorter* (in hops) than those operated by MPSFAP1.



(a) MPSFAP1, $a = 5, p = 0.2, p_v = 0.2$ (b) MRPSFAP, $a = 5, p = 0.2, p_v = 0.2$

Figure 7.11: Path additions and deletions for MPSPAF1 and MRPSFAP for demand matrix series M_2



(a) MPSFAP1, $a = 5, p = 0.2, p_v = 0.2$ (b) MRPSFAP, $a = 5, p = 0.2, p_v = 0.2$

Figure 7.12: Hop additions and deletions for MPSPAF1 and MRPSFAP for demand matrix series M_2

When the variations in the traffic matrix are less significative ($p_v = 0.2$), we see from Figures 7.9(a) and 7.10(a)[1] that the number of paths and hops stays almost unchanged for layouts

[1]No value in the chart means that the particular traffic matrix was infeasible. Accumulated values are not

obtained from both optimization objectives, except in early matrixes, when MRPSFAP used solutions with longer paths in order to avoid path and hops additions. However, from Figures 7.9(b) and 7.10(b) it can be seen that MPSFAP1 accumulates additions through the set of matrix changes, even when changes in the layout do not lead to better values in the objective function (i.e. optimality in the sense of MPSFAP1 problem). Again, from Figures 7.11(a), 7.11(b), 7.12(a) and 7.12(b) we observe that the activity of path and hops additions and deletions is higher for MPSFAP1. Here the difference is accentuated by the fact that an important number of those changes were not needed for the given traffic changes, and as such MRPSFAP guarantees that changes are done when they become unavoidable.

7.7. Extensions for Multiple Classes of Service

MRPSFAPQ 1
Minimum Reconfiguration, Path Set and Flow Allocation Problem for Multiple Classes of Service

Given:

$$\mathbf{A}, \mathbf{C}, \mathbf{D_{l,(t+1)}}, \mathbf{H_{(t)}}, \mathbf{\Theta}, \mathbf{\Delta}, \mathbf{W}$$

minimize :

$$\varphi \sum_{l=1}^{L} {}^{t}\mathbf{W_{(l)}} \cdot \mathbf{H_{l,(t+1)}} + \nu \sum_{l=1}^{L} {}^{t}\mathbf{W_{(l)}} \cdot \mathbf{S_{l,(t+1)}} \tag{7.15}$$

subject to:

$$\mathbf{X_{(t+1)}} = \mathbf{A} \cdot \mathbf{B_{(t+1)}} \cdot \mathbb{1}_L \leq \mathbf{C} \tag{7.16}$$

$$\mathbf{R} \cdot \mathbf{B_{(t+1)}} = \mathbf{D_{(t+1)}} \tag{7.17}$$

$$h^k_{q,l(t+1)} \sum_{i=1}^{M} \frac{\lambda a^k_{q,i}}{C_i - x_{i(t+1)}} \leq \theta_{q,l} \qquad l = 1, \ldots, L; k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.18}$$

$$b^k_{q,l(t+1)} \leq h^k_{q,l(t+1)} \delta_{q,l} \qquad l = 1, \ldots, L; k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.19}$$

$$(h^k_{q,l(t+1)} - h^k_{q,l(t)}) h^k_{q,l(t+1)} \leq s^k_{q,l(t+1)} \qquad l = 1, \ldots, L; k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.20}$$

$$h^k_{q,l(t+1)} \in \{0, 1\} \qquad l = 1, \ldots, L; k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.21}$$

$$s^k_{q,l(t+1)} \in \{0, 1\} \qquad l = 1, \ldots, L; k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.22}$$

$$b^k_{q,l(t+1)} \geq 0 \qquad l = 1, \ldots, L; k = 1, \ldots, K_q; q = 1, \ldots, Q \tag{7.23}$$

Recalling the model extensions for multiple classes of service in section 4.7, the problem of reconfiguration and layout complexity can be extended to a *Minimum Reconfiguration,*

affected.

Path Set and Flow Allocation Problem for Multiple Classes of Service (MRPSFAPQ). The demand matrix $\mathbf{D}$ defined in (4.27) considering multiple commodities for each node pair is now individualized in time: $\mathbf{D_{(t)}}$ is the demand matrix for instant t, and $\mathbf{D_{(t+1)}}$ is the demand matrix for calculating the new layout (instant $(t+1)$). Similarly, $\mathbf{B_{(t)}}$ as defined in (4.28) and $\mathbf{H_{(t)}}$, are the current path flow matrix and used path matrix, while $\mathbf{B_{(t+1)}}$ and $\mathbf{H_{(t+1)}}$ are the corresponding matrixes for the new layout at $(t+1)$. The classes of service are indexed $l = 1, \ldots, L$.

7.8. Conclusions

In this Chapter, the traffic dynamics have been considered. In this context, we find that for large networks and even when the MPLS layouts for successive demand matrixes are calculated according to the objectives defined for the MPSFAP problems, the reconfiguration complexity can be also very large, increasing the operational costs due to reconfiguration of the network. The reconfiguration costs can be even higher than the cost of operating the network in a sub-optimal state. The major contribution in this Chapter consist in the definition of a cost function for the optimization of the new layout, which has takes into account the currently used layout, in order to minimize complexity of the new layout as well as minimizing the reconfiguration complexity. The formulated problem is then referred as MRPSFAP (Minimum Reconfiguration and Path Set Flow Allocation Problem). The problem is formulated mathematically, as in the MPSFAP problems in Chapter 4, as a MINLP problem. Constraints are included so as to guarantee some QoS parameters as throughput and end-to-end path delay. Here again, the complexity of the problem formulated limits the tractable size of the problems by use of a deterministic solver.

In order to validate the model we resort to a deterministic solver, obtaining results for small network topologies. Results are obtained for two matrixes representing a mild to heavy load condition, from which a series of 24 traffic matrixes are generated. Results for the MRPSFAP problem are compared against results obtained by solving as if each traffic matrix were solved independently using the MPSFAP objective alone. Results show that we significatively reduce the number of hops and paths having to be changed from one layout to the next one as the traffic matrix evolves in time. In order to deal with the problem size limitation when using deterministic solvers, an heuristic based on a further adaptation of the MFD presented in Chapter 5 is developed in the next Chapter.

8. Contribution to the Development of Heuristics for Solving the Minimum Reconfiguration and Path Set Flow Allocation Problem (MPRSFAP)

8.1. Selecting Efficient Heuristics

As the MPSFAP problems, the MRPSFAP problem is also $\mathcal{NP}$-*complete*. In Chapter 7 the MRPSFAP problem was evaluated for small networks, which allowed us to to acquire some insight into the behavior of the objective function with respect to the MPSFAP objectives, and somewhat determine the interest of such optimization objectives for the design of MPLS layouts. However, heuristics are needed to solve large instances of the MRPSFAP problem due to the problem size limitations of the deterministic methods used in available solvers. In Chapter 5, meta-heuristics and ad-hoc heuristics for solving the dimensioning problems (MPSFAP) formulated in Chapter 4 were developed and evaluated. Algorithms implementing meta-heuristics based on Tabu Search methods (TS), as well as ad-hoc heuristics based on the Modified Flow Deviation method (MFD) were developed and tested both on small and large topologies. In Chapter 6, the implemented algorithms were evaluated for large networks under the same conditions (i.e. on the same hardware and operating environment), leading to the confirmation of the interest in including reconfiguration objectives together with dimensioning objectives for the design of MPLS layouts.

Analyzing the quality of the results obtained for the MPLS layout optimization of MPSFAP objectives on large networks, together with the corresponding computational effort they require, we conclude that the Tabu Search algorithm implemented is largely outperformed by the Modified Flow Deviation algorithm. Both, the quality of the results in terms of the value of the objective function, but mainly the computational performance of the MFD compared to the TS algorithms, encourage us to further develop ad-hoc heuristics in order to find a suitable and efficient algorithm to solve the MRPSFAP problem. In particular, the MFD algorithm implemented in section 5.5 in Chapter 5 already presents characteristics that make it naturally well adapted to efficiently solve the MRPSFAP problem with few modifications. In what follows, we present our contribution to the development of heuristics for solving the MRPSFAP problem, the *Modified Flow Deviation Algorithm for Reconfiguration (MFD-R)*, describing the modifications introduced to the MFD described.

8.2. Adapting the Modified Flow Deviation Algorithm (MFD) for Solving the MRPSFAP Problem

Recalling from section 5.5, the basic idea behind the MFD algorithm is to start from a *shortest path* layout in terms of number of hops, which can be *feasible* or *infeasible* with respect to the constraints (4.13),(4.14) and(4.15) defined in Chapter 4, and then shift flow on *one node pair at a time* in the direction of reducing the value of the objective function, which is defined as a function of the overload on the link (besides the weighted sum of the used paths). As such, decreasing the value of the objective function ensures that we shift flow also in the direction of converting the current solution into a feasible solution. The procedure of shifting flow on a node pair basis starting from a shortest path layout with single multiplicity (i.e. only one path connecting each node pair), ensures that the minimum quantity of paths (and shortest in quantity of hops) are added in the process, so when the layout obtained after a given iteration is *feasible*, it will likely be also the optimal in terms of MPSFAP1 objective. This has been shown when evaluating results for the MFD algorithm in section 5.5.6.

The algorithm finishes if no candidate is found to make a flow shift in the direction of decreasing the overload in the network or *shift direction*. If this happens and the constraints are still not met, we conclude that the problem is *infeasible*. However, we are not sure that the problem is infeasible if we have not tested all possible candidates in the set of paths. To improve the algorithm we introduce a modified version in Algorithm 5.6 to make appear all possible candidates before declaring the problem infeasible.

The MFD algorithm can be easily adapted to solve the MRPSFAP problem taking advantage of the particular characteristics of the above described procedure. The way we initialize the algorithm and the flow shift direction are chosen in a node pair basis *guarantee* that the solution found will be near-optimal in the sense of MPSFAP1 objective. However, the shift direction is searched among all the possible paths, what doesn't take into account the currently used paths. Indeed, a path that is not being used in the current layout can appear as a candidate to have flow shifted in, what would produce a path addition. We need to modify the way we chose the shift direction in order to explore first the paths that are being currently used and if no shift direction can be found, then all possible paths are explored. This will produce layouts with a low number of path additions, weighted by their lengths in hop-counts. However, it must be noted that no solution obtained through the MFD algorithm can be guaranteed to be globally optimal, since the order in which the flow shifts are realized affects the progression of the algorithm. We can control the order in which flow shifts are made by means of the *choice index* defined in Algorithm 5.11.

8.3. A Modified Flow Deviation Algorithm for Reconfiguration (MFD-R)

Formally, given the current operational layout $\mathbf{H_{(t)}}$ and the new demand matrix $\mathbf{D_{(t+1)}}$ (along with $\mathbf{A}$, $\mathbf{\Theta}$, $\mathbf{W}$ and $\mathbf{C}$), Algorithm 5.6 is modified to find the new set of used paths $\mathbf{H_{(t+1)}}$ and path flow vector $\mathbf{B_{(t+1)}}$.

Algorithm 8.1 includes the modifications introduced to the MFD algorithm, mainly regarding the way the set of paths in the arc-path incidence matrix $\mathbf{A}$ is explored. In the version of the algorithm to solve the MPSFAP1 problem, the whole matrix is explored on every iteration, since no reconfiguration objective is considered. When the reconfiguration objective is included, the way in which the set of paths is explored to find a shift direction which would produce a decrease in the value of the objective function becomes a key factor. If all paths in $\mathbf{A}$ are explored at each iteration as in the MFD algorithm, then paths that are not in $\mathbf{H_{(t)}}$ could result included in$\mathbf{H_{(t+1)}}$ (i.e. paths that are not used in the current layout) when searching for a shift direction. The set of paths to explore in a particular iteration of the algorithm is increased as no shift direction is found in the current set, until the whole matrix $\mathbf{A}$ has been searched or a shift direction found.

8.3.1. Initialization: The Initial Layout

The first modification consists in the way the initial layout is generated. One shortest path per node pair is found within the set of currently used paths $\mathbf{H_{(t)}}$ instead of within the whole matrix $\mathbf{A}$ as in MFD. This guarantees that the initial proposed solution exclusively uses paths in the set of paths that were already used in the current solution. Algorithm 8.2 calculates the single shortest path for every node pair within the set given as argument. In the initialization phase it is called with $\mathbf{H_{(t)}}$, so the returned set of shortest paths $\mathbf{P}$ contains paths only in the set $\mathbf{H_{(t)}}$. Later in the MFD-R algorithm, the procedure for calculating the set of single shortest paths is called with different set of paths to progressively include larger set of candidates if no shift direction could be found in sets containing already used paths.

Require: $\mathbf{F}, \mathbf{V}$ {$\mathbf{V}$ is the set of paths within which the shortest paths must be determined}
Ensure: $\mathbf{P}$
$\mathbf{P} = \left\{a_q^k : \min_k \left\{F_q^k\right\} \wedge a_q^k \in \mathbf{V}\right\}$
Return $\mathbf{P}$

Algorithm 8.2: Pseudo Code for Calculating the Set of Shortest Paths Within a Path Set

Require: $\mathbf{A}, \mathbf{\Theta}, \mathbf{W}, \mathbf{D_{(t+1)}}, \mathbf{C}, \mathbf{H_{(t)}}$ as defined in Chapter 7
Ensure: $\mathbf{B_{(t+1)}}, \mathbf{H_{(t+1)}}$ or INFEASIBLE
  $\mathbf{B_{(t+1)}} \leftarrow 0$
  $\mathbf{H_{(t+1)}} \leftarrow \varnothing$ {set of currently used paths}
  $\mathbf{F} \leftarrow \mathbf{W}$
  $\Delta\mathbf{B_{(t+1)}} \leftarrow 0$
  $\mathbf{P} \leftarrow \varnothing$ {set of current shortest paths}
  $\mathbf{P} \leftarrow$ **find_shortest_paths**$(\mathbf{F}, \mathbf{H_{(t)}})$
  for all $(a_q^k \in \mathbf{P})$ **do**
    $b^k_{q,(t+1)} \leftarrow d_{q,(t+1)}$ {assign all the demands to the shortest paths}
  end for
  $\mathbf{H_{(t+1)}} \leftarrow \mathbf{P}$
  $\mathbf{X_{(t+1)}} \leftarrow \mathbf{A} \cdot \mathbf{B_{(t+1)}}$
  $\mathbf{F} \leftarrow$ **update_path_weights**$(\mathbf{X_{(t+1)}})$
  $\mathbf{P} \leftarrow$ **find_shortest_paths**$(\mathbf{F}, \mathbf{H_{(t)}} \cup \mathbf{H_{(t+1)}})$
  $\mathbf{\Delta B_{(t+1)}} \leftarrow$ **find_shift_direction**$(\mathbf{B_{(t+1)}}, \mathbf{P}, \mathbf{H_{(t+1)}}, \mathbf{F})$
  if $(\mathbf{\Delta B_{(t+1)}} = 0)$ and (constraints are not met) **then**
    $\mathbf{\Delta B_{(t+1)}} \leftarrow$ **examine_thresholds**$(\mathbf{F}, \mathbf{H_{(t)}} \cup \mathbf{H_{(t+1)}}, \mathbf{B_{(t+1)}}, \mathbf{H_{(t+1)}})$
    if $(\mathbf{\Delta B_{(t+1)}} = 0)$ **then**
      $\mathbf{P} \leftarrow$ **find_shortest_paths**$(\mathbf{F}, \mathbf{A})$
      while $(\mathbf{\Delta B_{(t+1)}} = 0)$ **do**
        $\mathbf{\Delta B_{(t+1)}} \leftarrow$ **examine_thresholds**$(\mathbf{F}, \mathbf{A}, \mathbf{B_{(t+1)}}, \mathbf{H_{(t+1)}})$
      end while
    end if
  end if
  while $(\mathbf{\Delta B_{(t+1)}} \neq 0)$ **do**
    $\phi \leftarrow$ **find_shift_factor**$(\mathbf{\Delta B_{(t+1)}}, \mathbf{X_{(t+1)}})$
    $\mathbf{B_{(t+1)}} \leftarrow \mathbf{B_{(t+1)}} + \phi\mathbf{\Delta B_{(t+1)}}$
    $\mathbf{X_{(t+1)}} \leftarrow \mathbf{A} \cdot \mathbf{B_{(t+1)}}$
    $\mathbf{H_{(t+1)}} \leftarrow \cup_{k=1,\ldots,K_q;q=1,\ldots,Q} \left\{ a_q^k : b^k_{q,(t+1)} > 0 \right\}$
    $\mathbf{F} \leftarrow$ **update_path_weights**$(\mathbf{X_{(t+1)}})$
    $\mathbf{P} \leftarrow$ **find_shortest_paths**$(\mathbf{F}, \mathbf{H_{(t)}} \cup \mathbf{H_{(t+1)}})$
    $\mathbf{\Delta B_{(t+1)}} \leftarrow$ **find_shift_direction**$(\mathbf{B_{(t+1)}}, \mathbf{P}, \mathbf{H_{(t+1)}}, \mathbf{F})$
    if $(\mathbf{\Delta B_{(t+1)}} = 0)$ and (constraints are not met) **then**
      $\mathbf{\Delta B_{(t+1)}} \leftarrow$ **examine_thresholds**$(\mathbf{F}, \mathbf{H_{(t)}} \cup \mathbf{H_{(t+1)}}, \mathbf{B_{(t+1)}}, \mathbf{H_{(t+1)}})$
      if $(\mathbf{\Delta B_{(t+1)}} = 0)$ **then**
        $\mathbf{P} \leftarrow$ **find_shortest_paths**$(\mathbf{F}, \mathbf{A})$
        while $(\mathbf{\Delta B_{(t+1)}} = 0)$ **do**
          $\mathbf{\Delta B_{(t+1)}} \leftarrow$ **examine_thresholds**$(\mathbf{F}, \mathbf{A}, \mathbf{B_{(t+1)}}, \mathbf{H_{(t+1)}})$
        end while
      end if
    end if
  end while
  if constraints are met **then**
    Return $\mathbf{B_{(t+1)}}, \mathbf{H_{(t+1)}}$
  else
    Return INFEASIBLE
  end if

Algorithm 8.1: Pseudo-code for the Modified Flow Deviation Method Applied to the MRPSFAP Problem

8.3.2. Search for a Shift Direction: Progressive Exploration

Once the initial layout established, it can result feasible or unfeasible. If it is feasible (i.e. constraints are met) the algorithm has found a feasible solution with the minimum number of hops using paths that were already used in the previous solution. If the initial layout is infeasible, then the algorithm starts iterating until a feasible layout is found or no shift direction could be found (in which case we declare the problem infeasible). The main iteration in Algorithm 8.1 ensures a way of finding a shift direction that explores all the possible candidates by increasing the size of the set in which paths are searched for as no shift direction could be found in smaller sets. There are two possible sets to consider:

- The set of used paths in the current layout along with the set of currently used paths in the iteration: $\mathbf{H_{(t)}} \cup \mathbf{H_{(t+1)}}$.
- The whole arc-path incidence matrix $\mathbf{A}$.

As long as a shift direction can be found in the set $\mathbf{H_{(t)}} \cup \mathbf{H_{(t+1)}}$, we are sure that paths being included in $\mathbf{H_{(t+1)}}$ were already used in the layout at t or were already introduced in the current layout being calculated at $(t+1)$ because the shift direction had to be found looking in $\mathbf{A}$ in a previous iteration. At each iteration, the search is narrowed again, as the path weights are updated making possibly appear new candidates in the smaller set. Algorithm 8.3 examines all possible candidates within the set given as argument according to the improvement introduced in Algorithm 5.6.

Require: $\mathbf{F}, \mathbf{V}, \mathbf{B_{(t+1)}}, \mathbf{H_{(t+1)}}$ {$\mathbf{V}$ is the set of paths to explore for a shift direction}
Ensure: $\mathbf{\Delta B_{(t+1)}} > 0$ if a possible shift direction exists, and 0 otherwise.
  $\mathbf{\Delta B_{(t+1)}} \leftarrow 0$
  $threshold \leftarrow 1$
  $\mathbf{F_b} \leftarrow \mathbf{F}$
  while $(\mathbf{\Delta B_{(t+1)}} = 0)$ and $(threshold < network\_diameter)$ **do**
    $\mathbf{F_b} \leftarrow$ **increase_threshold_level**$(\mathbf{F_b}, threshold)$
    $\mathbf{P} \leftarrow$ **find_shortest_paths**$(\mathbf{F}, \mathbf{V})$
    $\mathbf{\Delta B_{(t+1)}} \leftarrow$ **find_shift_direction**$(\mathbf{B_{(t+1)}}, \mathbf{P}, \mathbf{H_{(t+1)}}, \mathbf{F_b})$
    $threshold \leftarrow threshold + 1$
  end while
  Return $\mathbf{\Delta B_{(t+1)}}$

Algorithm 8.3: Pseudo Code for Examining All Possible Candidates for a Shift Direction Within a Given Set of Paths

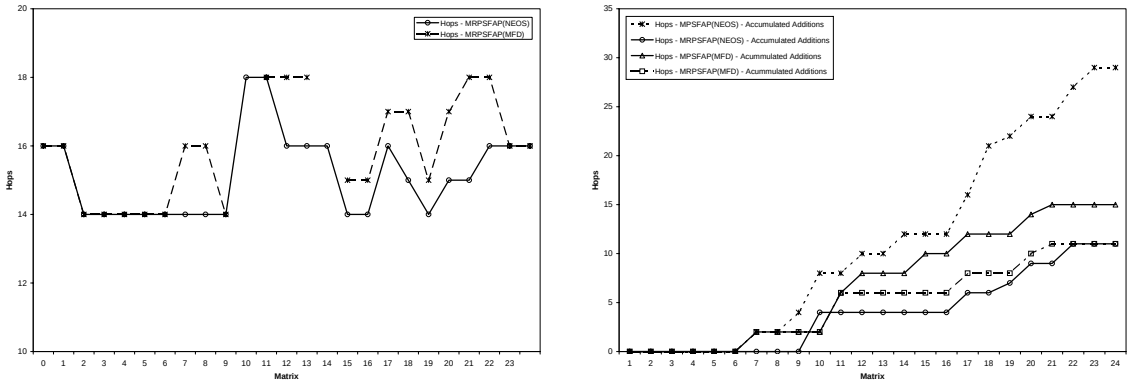
The procedures to calculate the *shift direction*, the *shift factor* and the *choice index*, defined in Algorithms 5.9, 5.10 and 5.11 stay the same as defined in Chapter 5, except that they are called with the instances at $(t+1)$ of the path flow vector $\mathbf{B_{(t+1)}}$ and used path matrix $\mathbf{H_{(t+1)}}$. All the other algorithms defined in section 5.5 do not need to be modified either, and are used as before.

8.4. Evaluation of the MFD-R Algorithm for MRPSFAP

In this section we first evaluate the Modified Flow Deviation for Reconfiguration (MFD-R) heuristic by comparing the results obtained from the MINLP deterministic solver for a small network topology (NET2) in Chapter 7, to the results obtained from the MFD-R algorithm for the set of matrixes M_1. Second, we obtain results for large network topologies such as VTHD and NSF presented in Chapter 6, using the MFD-R algorithm for two set of matrixes M_1^{VTHD} and M_1^{NSF}.

8.4.1. Evaluation of the MFD-R Algorithm on Small Networks

In order to evaluate the MFD-R algorithm, we compare the results obtained for NET2 network topology of previous chapters by using both, the MFD-R and the deterministic solver (MINLP). As before, we use a small topology to evaluate the algorithm due to the problem size limitations of the deterministic solver. The same set of matrixes M_1 of Chapter 7 is used to compare results for MFD-R and MINLP solver, and all runs for MFD-R are made under the same conditions on the same machine. Figure 8.1(a) shows the total number of hops required by MRPSFAP by using both MFD-R and the MINLP solver.



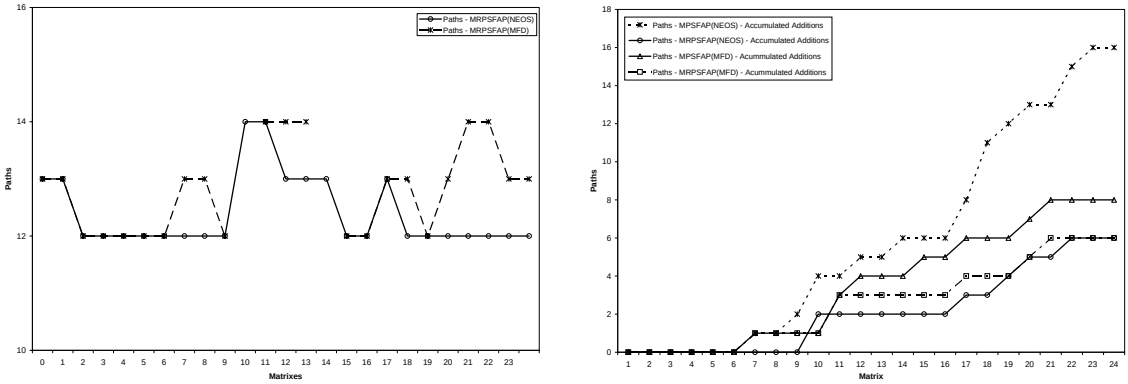
(a) Active Hops

(b) Accumulated Hop Additions

Figure 8.1: Used hops and accumulated hop additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1 using MINLP Solver (NEOS) and MFD-R Heuristic

From the figure, we see that the number of hops required when solving the MRPSFAP problem with MFD is greater than the number of hops required when solving the same instance with the deterministic solver. From Figure 8.2(a) we observe the same trend for the number of paths. Not surprisingly, this conclusion for the MFD-R algorithm (observed also for the MFD algorithm applied to the MPSFAP problems) was somewhat expected. In fact, both the MFD and MFD-R algorithms are heuristics to find approximate solutions to a complex problems, tackling the size limitations imposed by the deterministic solvers.

From Figures 8.1(b) and 8.2(b) we see that the number of accumulated additions (hops and paths) when optimizing the layouts for the MPSFAP objective by means of the MINLP solver is significantly higher than the values obtained by using the MFD algorithm for the same objective.



(a) Active Paths

(b) Accumulated Path Additions

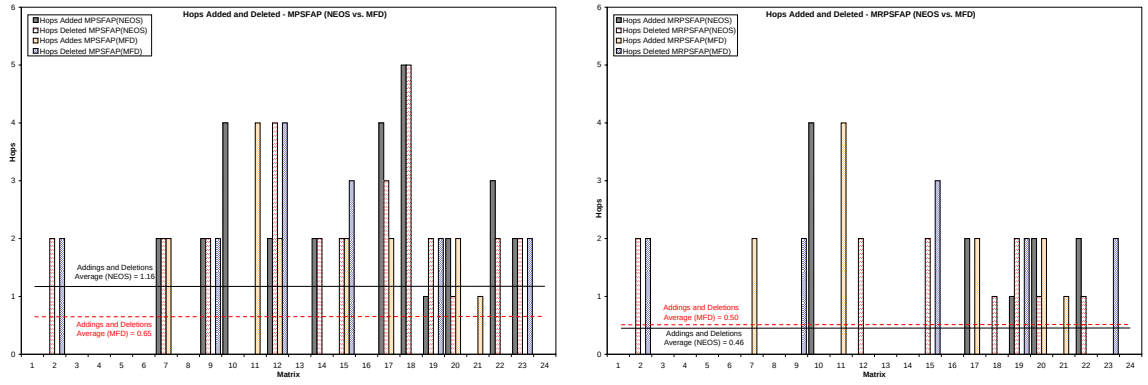
Figure 8.2: Used paths and accumulated path additions for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1 using MINLP Solver (NEOS) and MFD-R Heuristic

This is due to the fact that the heuristic in which is based the MFD algorithm produces already a reduced set of paths that is likely the one to be produced again when solving for the new demand matrix, even when the objective function does not consider the previously used set of paths for reconfiguration optimization. This is a new interesting conclusion about the MFD algorithm. Regarding the MFD-R algorithm, which takes into account the previously used set of paths to optimize both complexity of the new layout as well as reconfiguration complexity, results from the above mentioned figures also show that the MFD-R algorithm further improves on the results obtained by the MFD algorithm. In fact, the number of hop and path additions is further reduced with respect to the MFD results. More, as we would expect, the results obtained for the MRPSFAP objective through the deterministic solver are better than the ones obtained by MFD-R. All observed results allow us to conclude that the MFD-R algorithm is suitable to solve the MRPSFAP problems within a reasonable approximation, as shown in (8.1) from averages of the objective function obtained by both solvers.

$$f(s)\%_{MFD-R} = \frac{f(s)_{MFD-R} - f(s)_{MINLP}}{f(s)_{MINLP}} 100 = 5.33\% \tag{8.1}$$

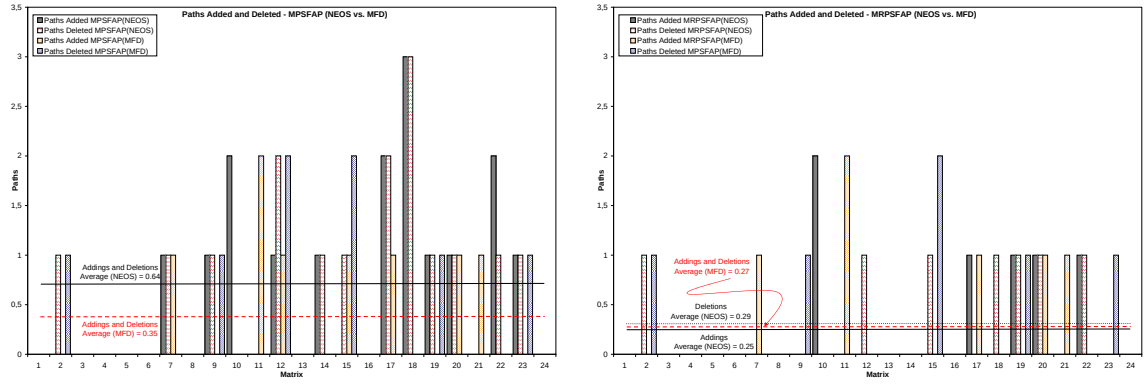
Figures 8.3 and 8.4 show the activity of added and deleted hops and paths respectively through the process of optimizing the matrix series M_1. We clearly see that the activity of hops additions and deletions for the MPSFAP1 objective is always higher than the same

activity for the MRPSFAP objective. Regarding the performance of the MFD-R algorithm as compared to the deterministic solver, we see that the average figures for additions and deletions for the MFD-R algorithm are only slightly higher than the ones observed for the MINLP solver.



(a) MPSFAP1 (b) MRPSFAP

Figure 8.3: Hops added and deleted for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1 using MINLP Solver (NEOS) and MFD-R Heuristic



(a) MPSFAP1 (b) MRPSFAP

Figure 8.4: Paths added and deleted for MPSFAP1 and MRPSFAP on NET2 for demand matrix series M_1 using MINLP Solver (NEOS) and MFD-R Heuristic

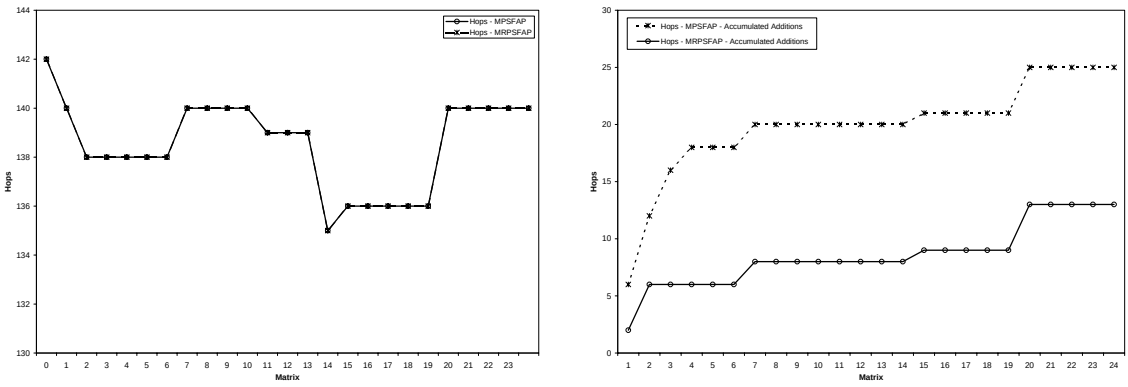
The average for additions and deletions (hops and paths) when using MFD as opposed to MFD-R confirm the observation already made for Figures 8.1 and 8.2 above. The averages for MFD are higher than the ones for MFD-R, but the difference with the MINLP solver when optimizing the MPSFAP objective alone are significantly higher, which shows that MFD is an already suitable algorithm to reduce reconfiguration complexity, even when it

does not take into account the previously used set of paths.

In next section we compare results from the MFD-R algorithm with results from the MFD algorithm for large networks. The conclusions we made in this section regarding the good performance of MFD for the reconfiguration objectives, even when the complexity alone was being minimized, will be useful when studying the comparative performance of MFD-R.

8.4.2. Results on Reconfiguration from the MFD-R Algorithm on Large Networks

In this section we compare results obtained by using MFD and MFD-R algorithms to solve the MPSFAP1 and MRPSFAP problems respectively on large networks. The large network topologies used in this section are the VTHD and NSF networks presented in Chapter 6. A set of demand matrixes conforming matrix series obtained as described in section 7.6.1 are presented to each topology. We denote by M_1^{VTHD} the matrix series presented to the VTHD network, and M_1^{NSF} the matrix series presented to the NSF network.



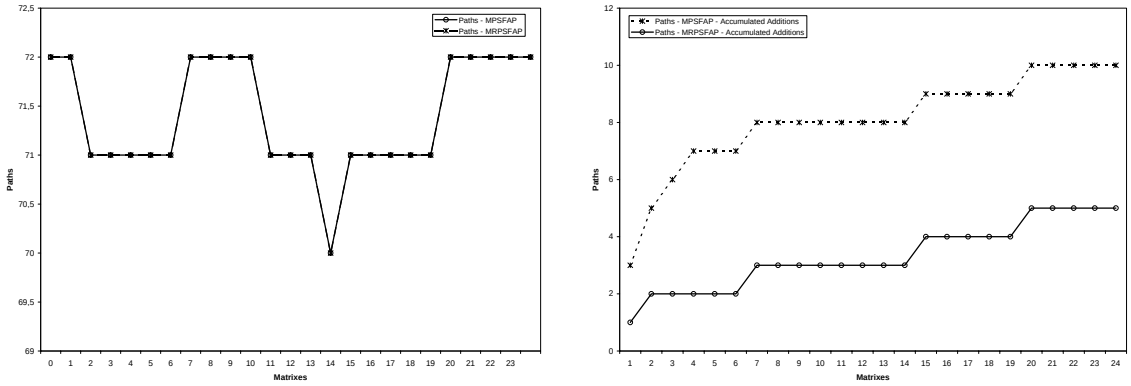
(a) Active Hops

(b) Accumulated Hop Additions

Figure 8.5: Active hops and accumulated hop additions for MPSFAP1 and MRPSFAP on VTHD for demand matrix series M_1^{VTHD} using MFD and MFD-R heuristics

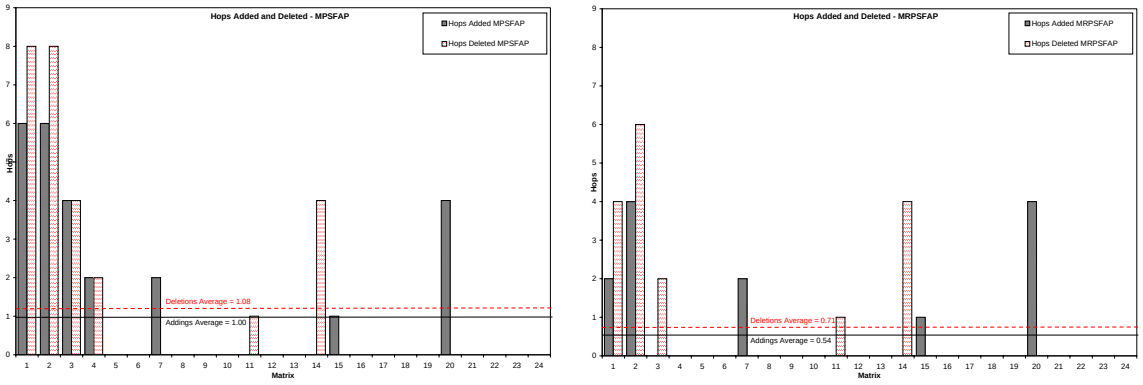
Figures 8.5(a) and 8.9(a) show the total number of active hops for each of the matrixes in the corresponding series for VTHD and NSF topologies, when solving each of the matrixes for the MPSFAP1 objective and the MRPSFAP objective. Figures 8.6(a) and 8.10(a) show the corresponding values for the total number of used paths. We can observe for VTHD that exactly the same number of hops and paths is used when solving either for the objective MPSFAP1 or MRPSFAP. Looking at Figures 8.5(b) and 8.6(b) we can see that the number of accumulated hop and path additions is higher when solving the matrix series for the objective MPSFAP1 (algorithm MFD) than for the objective $MRPSFAP$ (algorithm MFD-R), although the difference is not much significant. Looking however to the quantity of

active hops and paths for NSF, we can see that the number of hops required by MRPSFAP is generally higher than the number of hops required by MPSFAP1, while the number of paths required by MRPSFAP is always lower than the number required by MPSFAP1.



(a) Active Paths (b) Accumulated Path Additions

Figure 8.6: Active paths and accumulated path additions for MPSFAP1 and MRPSFAP on VTHD for demand matrix series M_1^{VTHD} using MFD and MFD-R heuristics

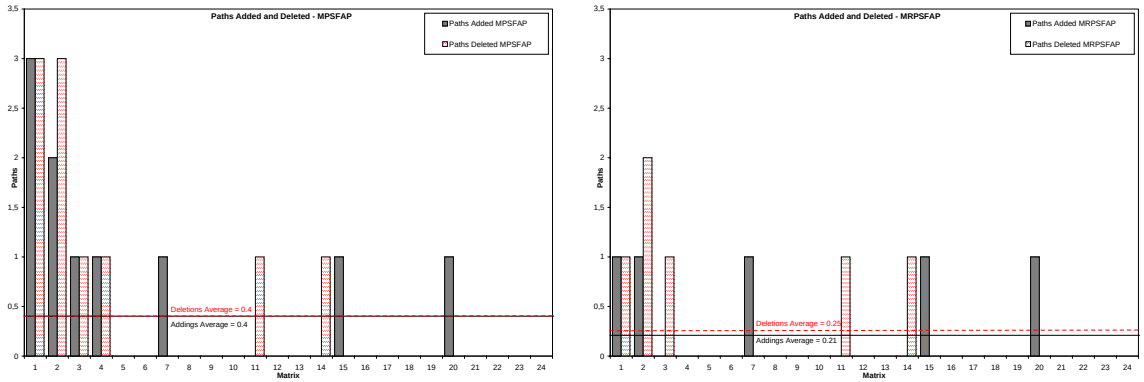


(a) MPSFAP1 (b) MRPSFAP

Figure 8.7: Hops added and deleted for MPSFAP1 and MRPSFAP on VTHD for demand matrix series M_1^{VTHD} using MINLP Solver (NEOS) and MFD-R heuristics

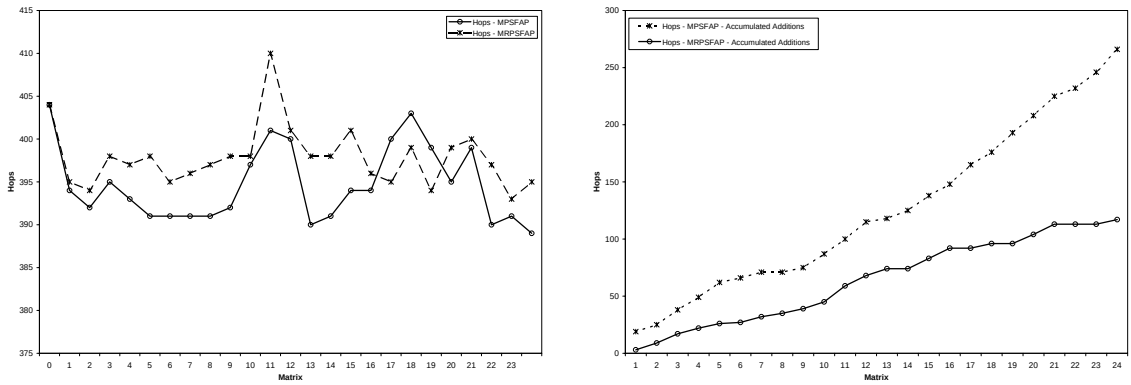
This is a somewhat expected result, since the MFD-R heuristic concentrate in finding a feasible solution within the set of already used paths first, and then it looks for new paths only when no shift direction could be produced within the previously used set. This produces layouts with a set of paths likely to be a subset of the previously used set for the MRPSFAP objective, paying the cost of longer paths in order to reduce path additions with respect to the previous set, while the MPSFAP1 objective is free to chose the shortest paths at every

iteration without any reference to the past.



(a) MPSFAP1 (b) MRPSFAP

Figure 8.8: Paths added and deleted for MPSFAP1 and MRPSFAP on VTHD for demand matrix series M_1^{VTHD} using MINLP Solver (NEOS) and MFD-R heuristics

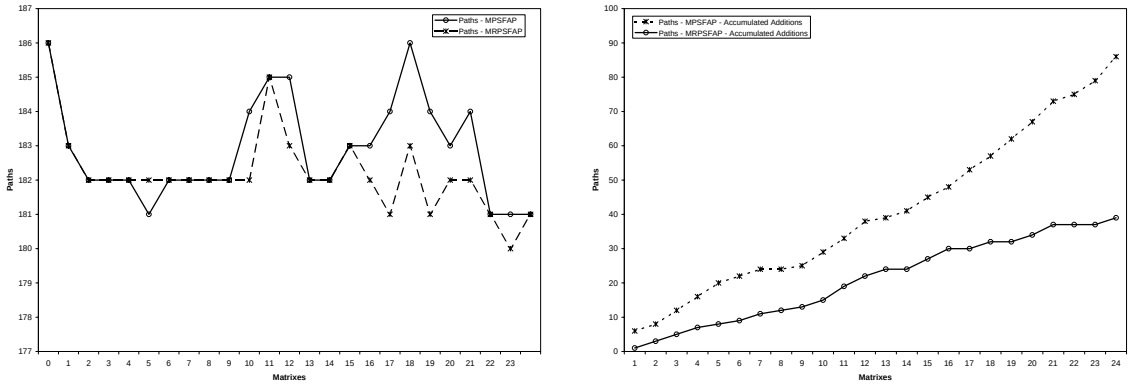


(a) Active Hops (b) Accumulated Hop Additions

Figure 8.9: Active hops and accumulated hop additions for MPSFAP1 and MRPSFAP on NSF for demand matrix series M_1^{NSF} using MFD and MFD-R heuristics

The number of accumulated hop and path additions for NSF in Figures 8.9(b) and 8.10(b) show a significant difference with the values observed for VTHD. The number of accumulated hop and path additions for the MRSPFAP objective in the case of NSF results almost doubled with respect to the same values for the MPSFAP1 objective. In fact, this can be explained by the nature of the topologies considered. VTHD topology doesn't offer as many possible path combinations to connect a given node pair as NSF does, presenting critical links (e.g. links 12, 24, 9, 21, 11, 23, 10 and 22 in Figure 6.1) that are used by all paths connecting some node pairs (e.g. nodes 5, 4 and 9 in the same figure). The impact of

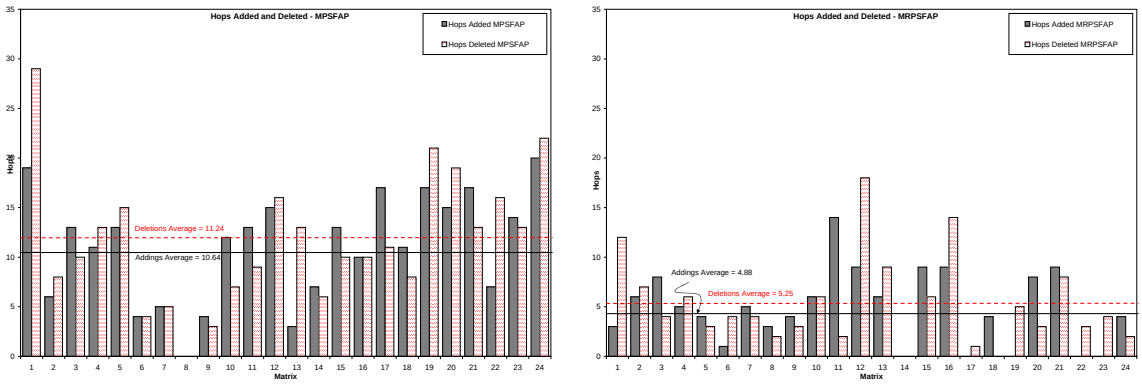
using a MRPSFAP objective to reduce reconfiguration and layout complexity is then, as we would expect, strongly dependent on the network topology, but from observed results we can conclude that reconfiguration and complexity are always reduced by using the MRPSFAP objective instead of MPSFAP1 objective (i.e. solving always by MFD-R algorithm).



(a) Active Paths

(b) Accumulated Path Additions

Figure 8.10: Active paths and accumulated path additions for MPSFAP1 and MRPSFAP on NSF for demand matrix series M_1^{NSF} using MFD and MFD-R heuristics



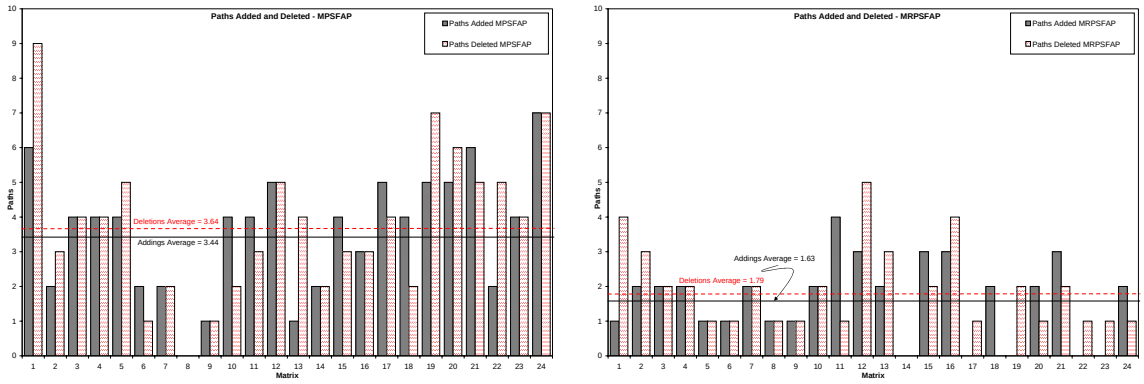
(a) MPSFAP1

(b) MRPSFAP

Figure 8.11: Hops added and deleted for MPSFAP1 and MRPSFAP on NSF for demand matrix series M_1^{NSF} using MINLP Solver (NEOS) and MFD-R heuristics

Figures 8.7 and Figure 8.8 show the hop and path adding and deletion activity through the optimization of the matrixes belonging to the series M_1^{VTHD} for the VTHD network topology. Figures 8.11 and 8.12 show the corresponding values for the NSF network. As observed already for the accumulated additions for both networks, the observed activity for VTHD network is significantly lower than the one observed for NSF network relative to the

size of the corresponding networks.



(a) MPSFAP1 (b) MRPSFAP

Figure 8.12: Paths added and deleted for MPSFAP1 and MRPSFAP on NSF for demand matrix series M_1^{NSF} using MINLP Solver (NEOS) and MFD-R heuristics

It can also be observed that the additions and deletions are near symmetrical for VTHD network, while they exhibit a greater asymmetry for the NSF network, the average of deletions being greater than the average of additions for both paths and hops, which is a wanted result. Finally, even when the average values for additions and deletions for VTHD network are relatively low, the ratio from values obtained for MPSFAP1 objective to the values obtained for MRPSFAP objective is almost the same than for NSF network (i.e. the averages for MRPSFAP are nearly half of the averages for MPSFAP1), which drives us to conclude that the MFD-R algorithm performs consistently well to optimize the MRPSFAP objective. Observed execution times for MFD-R are within the same order than the execution times observed for MFD under the same conditions.

8.5. Conclusions

The major contribution in the the present Chapter consists in the development of an heuristic method based on the MFD algorithm presented in Chapter 5. The MFD algorithm has been shown to perform well both in the quality of the results obtained and in the execution times needed to calculate the layouts. Implementation of the algorithm has been completed, and further evaluation is presented in this chapter. Results obtained on small networks allow us to conclude that the MFD-R algorithm performs within a reasonable range from the optimal results obtained by using a deterministic solver (MINLP), and important conclusions were obtained about the behavior of the MFD and MFD-R algorithms. Finally, the MFD-R was evaluated for large networks, and results compared to the results obtained with the MFD algorithm for the same traffic matrix series. The MFD-R algorithm has shown to

perform within our expectations for optimizing the MRPSFAP objective of reconfiguration and layout complexity, and as a consequence, obtained results allows us to foresee practical applications of the MFD-R algorithm in an operational context. Also, comparison to results obtained with the MFD algorithm helped us to acquire further insight in the behavior of both objectives. As a final and important conclusion, the MFD-R algorithm can be used in practical operational environments to calculate layouts with dramatic improvement under dynamic traffic situations both in the sense of layout complexity and reconfiguration complexity.

General Conclusions

This Chapter presents the general conclusions inferred from the studies described in the present document. We summarize the main contributions before giving the future research directions that stem from this work.

Contributions

This thesis concentrates on the traffic engineering mechanisms involved in the dynamic control of the network behavior considering long-term timescales. We locate our work in the technological context of Next Generation IP Networks, where a unified transport based on IP protocol is envisaged. As such, and to allow the operator to provide QoS guarantees while optimizing the network resources traffic engineering mechanisms have to be added to IP transport functionalities. Modelling the TE system as a closed loop control system, three main control loops can be identified, each loop corresponding to a timescale. TE mechanisms inscribed within a particular timescale aim at helping to control the network behavior in order to ensure that the main objectives of TE are met: to provide QoS guarantees while ensuring an efficient use of network resources. In the context of NG-IP networks, MPLS allows for the implementation of evolved TE mechanisms by providing the possibility of establishing source routed paths. In the long-term timescales, the TE tasks are oriented to planning and dimensioning of the network according both to underwritten service contracts and to business objectives. From an operator's point of view, the business objectives are realized in a significative part through OPEX and CAPEX reduction. The contributions of this thesis are then mainly oriented in this sense, where we propose realistic objectives to dimension the network according to the business objectives, while meeting the user oriented objectives (i.e. QoS guarantees). Dimensioning the network in a dynamic context involves the reconfiguration aspects, as it is not enough to obtain an optimal layout for a new traffic matrix if the transition from the old layout to the new layout is too complex that no operational cost reduction is obtained in the process.

Dimensioning of MPLS Layouts. Once the network deployed, the cost of operation is related to the complexity of the network *layout*. In a connected oriented transport paradigm, as we assume to be the framework in the present work, the layout is represented by the set of paths connecting each node pair and the flows to be allocated to each path in order to meet the demands. We consider that the complexity of this *layout*, and in consequence the

cost of operation, is related to the total number of paths weighted by the quantity of hops each path is using. The larger the network, the more important this cost becomes. The QoS guarantees such as end-to-end path delay and throughput for a given node pair can be guaranteed through constraints, while the actual cost when calculating the optimal layout for a given traffic matrix is represented by the number of length-weighted paths the layout requires to route a given demand. In this sense, the MPSFAP problems are formulated to optimize the layouts with respect to the quantity of total hops needed to route such demand. To our knowledge, no such optimization objectives have been considered so far in the context of MPLS and IP networks. Considerable work has been carried out in the field of optical networks and in the field of general optimal routing in IP networks considering other performance objectives such as minimizing congestion. The proposed problem formulation include extensions for multiple classes of service. Results for both small and large topologies show that average path *multiplicity* (i.e. the quantity of paths required to transport the demand between any given node pair) is close to 1 if demands are a fraction of total available capacity connecting every node pair.

Reconfiguration of MPLS Layouts. When considering the traffic dynamics, the control loop at long-term timescales will act to drive the network to an optimal operational state for the new demand matrix inferred from observation of the network state. For large networks, the transition to the new optimal layout, even when optimized in terms of complexity according to the MPSFAP objectives, can be as complex as a non optimized layout. In other words, the complexity of the transition from an optimized layout to the next optimized layout may result so complex that the incurred cost of *reconfiguration* results higher than the cost of operating the suboptimal layout (penalties for not meeting the service contracts considered). The costs of reconfiguration are relatively high as reconfiguring a large operational network may need a large amount of spare network resources to avoid service disruption times to occur. If such resources are not made available, such complex reconfiguration may take longer times to be set up, and service contracts may be unhonored. Considering reconfiguration complexity objectives along with layout complexity objectives seems then natural in a NG-IP technological framework when considering more realistic factors as the traffic dynamics. The resulting problem we formulate, the MRPSFAP problem, includes both objectives: layout complexity as well as reconfiguration complexity by including in the objective the total number of length-weighted paths together with the number of length-weighted path additions, while keeping the QoS guarantees for the nex calculated layout as in the MPSFAP problems. As in the case of dimensioning, extensions for multiple classes of service are included. To our knowledge, this problem has not been extensively studied, besides some initial studies in the field of optical networks. Results obtained for small network topologies already show that savings in terms of quantity of paths being set up and teared down at every reconfiguration stage are significant, allowing the operator to make transition

the network from the current optimal operational layout to the next one at a minimum cost.

Heuristics for Solving the Dimensioning Problem. The dimensioning problem, referred to as Minimum Path Set and Flow Allocation Problem (MPSFAP) in the present work and formulated as MINLP multicommodity flow problem, is $\mathcal{NP}$-*complete*. This imposes problem size limitations to the network topologies that can be solved with deterministic solvers, restricting the tractable solutions to a reduced set of unpractical network topologies. In order to tackle this problem size limitation, and to be able to apply the MPSFAP cost functions to the design of real-life size networks, approximated algorithms were developed based on both *generic* and *ad-hoc* heuristic methods. The generic heuristic makes use of Tabu Search methods to walk the solution space, while the ad-hoc heuristic method makes use of the knowledge we have on the particular problem being treated. The former has the advantage of being easily adaptable to solve new problems if the cost function is changed to reflect new operational cost and business models, the later is designed particularly for the cost function being treated. However, the gain in performance both in terms of quality of results and execution time largely justifies the efforts in developing and implementing the ad-hoc heuristics, as verified when observing the results obtained using both algorithms. Studying the well known flow deviation method to solve the multicommodity flow problem for convex objective functions [64, 31], we observed that the general procedure could be adapted to guarantee finding a minimum set of paths meeting the QoS constraints. The Modified Flow Deviation Algorithm has been then defined and implemented, outperforming the TS algorithm. The idea behind developing an ad-hoc heuristic instead of adapting the problem in order to be able to solve it through deterministic solvers or more general heuristics is to obtain an algorithm simple and quick enough capable of being implemented in a control loop in operational networks.

Heuristics for Solving the Reconfiguration Problem. As with the MPSFAP problem, the MRPSFAP problem is $\mathcal{NP}$-*complete*. Heuristics are needed in order to overcome the problem size limitation. Performance obtained with the TS algorithm compared to the ones obtained with the MFD algorithm for the MPSFAP problem no longer justify further development in that direction. We oriented our work towards an heuristic based on the same principles that the MFD algorithm, but restricting the search for a solution to the set of previously used paths as much as possible. The algorithm has been fully defined and described in the present work, and is currently under development and testing.

Future Directions

The study of the dimensioning and reconfiguration problems for MPLS layouts open many research directions as well as several implementation issues that we consider interesting for

further development and study.

Implementation. The possibility of having implemented algorithms capable of solving the dimensioning and reconfiguration problems in times compatible with the network operation and within reasonable tolerances from the optimal values when analyzed for small network topologies, attracts the interest of studying the behavior of actual operational networks when adapting to dynamic load conditions in long timescales. A very promising platform for studying such aspects is the VTHD french research network, a very high speed network already implementing MPLS and with a national span. The algorithm implementing the MFD heuristics for the dimensioning and reconfiguration problems can be implemented as the core of a long term control loop acting at a global scope in the network. An interface to the measurement and observation system can be implemented by reading the MIBS, and complementary by using COPS or a like protocol to launch active and passive measurements in the network. An inference system from those observations is needed in order to elaborate a demand matrix integrating the observations at the appropriate timescale. This demand matrix is used as the input to the reconfiguration algorithm, which takes into account the already deployed layout in order to minimize the reconfiguration complexity. The *differential* layout is then *communicated* to the management system through the change in labels and routes for the MPLS signaling protocols to actually configure the new layout. A new research direction arises when considering the actual implementation of the new layout, even when calculated to reduce the complexity of reconfiguration with respect to the currently used layout: the way in which the new paths are going to be added and some of the old paths are going to be teared down. A study on progressive techniques to reconfigure the MPLS layout constitutes a new and interesting direction to take.

Modelling of Demands. The traffic models used through this work are simplified models at long timescales of ingress traffic processes. As research on IP metrology goes on, we see the interest of taking those more evolved traffic models into account when optimizing the MPLS layout design. Also, modelling the traffic dynamics is important to the reconfiguration phase. To model the traffic dynamics at long timescales is necessary to correctly integrate the measures performed over a long period in order to infer a new traffic matrix predicting the traffic variations over the next period. Studies on dimensioning the network at shorter timescales is also estimated as an interesting research direction.

Impact on the Inner Control Loops. The TE mechanisms used at the different timescales are not independent one to another. The interaction among all the TE mechanisms have to be accounted for when deciding which TE mechanisms is to be used for which situation and under what conditions. In current management architectures, it is to the traffic engineering to decide on such matters. It would be interesting in the context of

next generation IP networks to take such decisions automatically, based on measurements and inference of the network state form those measures. However, it is first necessary to understand the interaction among all the TE mechanisms at the different planes of service integration in order to be able to infer rules helping to control the network behavior under almost any condition.

Impact on QoS: Traffic Separation. The problems of dimensioning and reconfiguration have been formulated taking into account extensions to support multiple classes of service, providing QoS guarantees to each class on a node pair basis. The complexity of the problem increases with the number of classes, as using label inferred LSPs for instance, requires a single path for every class of service for each node pair, so QoS guarantees can be provided individually. Even when extending the proposed algorithms to support multiple classes of service is quite natural and relatively straightforward, new factors have to be taken into account when calculating the optimal traffic distribution over the set of paths. Indeed, the cost functions may be have to be adapted to consider the relative differences in cost of multiplexing the different classes of service onto the available paths in the network adequately (e.g. a proportion of bandwidth dedicated to stream flows, and other part of traffic dedicated to elastic traffic, separation of paths for elastic traffic and for stream traffic).

Appendices

A. Multicommodity Flow Problems

A.1. Introduction

In the context of telecommunications networks, a commodity is a quantity of flow that we need to send from one node in the network (source) to another node in the network (destination). We want this flow to be sent along an optimal route, for instance, along the shortest path according to a cost function determining the path lengths. In many telecommunications applications, we want to transport several *commodities* at the same time, sharing the network resources and each commodity governed by its own network flow constraints. If the commodities do not interact in any way, then to solve routing problems with several commodities, we would solve each single-commodity problem separately. However, when the commodities share common facilities, the individual single-commodity problems are not independent, so to find an optimal flow we need to solve all the coordinated problems at the same time. The later is referred as a *multicommodity flow problem* [14].

In what follows, we present the *node* and *path* formulations of the multicommodity flow problem.

A.2. Node Formulation

Let x_{ij}^k denote the flow commodity k on arc (i,j), and let x^k and c^k denote the flow vector and unit cost vector for commodity k. Using this notation, we can formulate the multicommodity flow problem as:

$$\text{Minimize} \sum_{1 \leq k \leq K} c^k x^k \tag{A.1}$$

subject to:

$$\sum_{1 \leq k \leq K} w_{ij}^k \leq u_{ij} \qquad \text{for all } (i,j) \in A \tag{A.2}$$

$$\mathcal{N} x^k = b^k \qquad \text{for } k = 1, 2, \ldots, K \tag{A.3}$$

$$0 \leq x_{ij}^k \leq u_{ij}^k \qquad \text{for all } (i,j) \in A \text{ and all } k = 1, 2, \ldots, K \tag{A.4}$$

where A is the set of links in the network, u_{ij} is the arc capacity limiting the total flow of all commodities on that arc, and $\mathcal{N}$ is a *node-arc* incidence matrix of dimension $n \times m$ (i.e n nodes and m arcs). The constraints (A.3) are the *mass balance* constraints, modelling the flow of each commodity $k = 1, 2, \ldots, K$. The *bundle* constraints (A.2) tie together the commodities by restricting the total flow on each arc (i, j) to at most u_{ij}. Individual flow bounds u_{ij}^k on the flow of commodity k on arc (i, j) are also included, although most applications do not impose such constraints and can be set to $+\infty$.

A.3. Path Formulation

Let's consider a special case of the multicommodity flow problem where each commodity k has a single source node s^k and a single sink node t^k, and a flow requirement (demand) of d^k units between these source and sink nodes. No flow bounds are imposed on the individual commodities, other than the bundle constraints. We can then reformulate the multicommodity flow problem using path and cycle flows instead of link flows. Let's assume that the cost of every cycle $\mathbf{W}$ in the underlying network is nonnegative (e.g. if all arc flow costs are nonnegative). Imposing the nonnegativity condition on the cycle cost, implies that in some optimal solution to the problem, the flow on every cycle is zero, so we can eliminate the cycle flow variables. As a consequence, we can express any potentially optimal solution as the sum of flows on directed paths.

For each commodity k, let $\mathbf{P^k}$ denote the collection of all directed paths from the source s^k to the sink node t^k in the underlying network. Let $\delta_{ij}(P)$ be an arc-path indicator variable (i.e. $\delta_{ij}(P)$ is 1 if arc (i, j) is contained in the path P, and 0 otherwise). The flow decomposition theorem of network flows [14] states that some optimal arc flow x_{ij}^k can be always decomposed into path flows $f(P)$ by:

$$x_{ij}^k = \sum_{P \in \mathbf{P^k}} \delta_{ij}(P) f(P) \tag{A.5}$$

Let $c^k(P) = \sum_{(i,j)\in A} c_{ij}^k \delta_{ij}(P) = \sum_{(i,j)\in P} c_{ij}^k$ be the per unit cost of transporting flow on the path $P \in \mathbf{P^k}$ with respect to the commodity k. By substituting the path variables in the multicommodity flow formulation, we obtain the equivalent path flow formulation of the problem:

$$\text{Minimize} \sum_{1 \le k \le K} \sum_{P \in \mathbf{P^k}} c^k(P) f(P) \tag{A.6}$$

subject to:

$$\sum_{1\leq k\leq K}\sum_{P\in\mathbf{P^k}}\delta_{ij}(P)f(P)\leq u_{ij} \qquad \text{for all } (i,j)\in A \tag{A.7}$$

$$\sum_{P\in\mathbf{P^k}} f(P)=d^k \qquad \text{for } k=1,2,\ldots,K \tag{A.8}$$

$$f(P)\geq 0 \qquad \text{for all } k=1,2,\ldots,K \text{ and all } P\in\mathbf{P^k} \tag{A.9}$$

The path flow formulation of the multicommodity flow problem has a very simple constraint structure, constituting one of the reasons why we have chosen this formulation for our MPSFAP and MRPSFAP problems. The problem has a single constraint (A.7) for each arc (i,j) limiting the total flow (i.e. the sum of path flows) traversing that arc to at most its capacity u_{ij}. Moreover, the problem has a single constraint (A.8) for each commodity k expressing that the set of paths connecting the source node s^k to the destination node t^k transport a quantity of flow equivalent to the demand d^k for that commodity.

For a network with n nodes and m arcs and K commodities, the path flow formulation contains $m+K$ constraints. The arc formulation on the other hand contains $m+nK$ constraints (since it has one mass balance constraint for every node and commodity combination). This produces a problem with much less constraints for the path formulation than for the arc formulation. However, the difference is even more pronounced regarding the resolution methods that can be applied to either formulation. The savings in the number of constraints comes at a cost, since the path flow formulation has a variable for every path connecting a source and a sink nodes for each of the commodities. The number of variables will then grow very large (exponentially) with the size of the network. However, we expect that only few of the paths will actually carry flow in the optimal solution, and so the total number of variables will get drastically reduced.

A.4. Solution Approaches

Several approaches have been developed for solving the multicommodity flow problem, including [14]:

- Price-directive decomposition.
- Resource-directive decomposition.
- Partitioning methods.

Price-directive decomposition methods place Lagrangian multipliers (or prices) on the bundle constraints, bringing them into the objective function so that the resulting problem

gets decomposed int a separate minimum cost flow problem for each commodity k. These methods attempt to find appropriate prices so that some optimal solution to the resulting Lagrangian subproblem also solves the overall multicommodity flow problem. Dantzig-Wolfe decomposition is another approach to finding the prices for the de decomposed problem. We note that the multicommodity problem formulated have a set of *easy* constraints (the flow constraints) and a set of *complicating* constraints (the bundle constraints). The approach begins like Lagrangian relaxation by ignoring or imposing prices on the bundle constraints and solving Lagrangian subproblems with only the single network flow constraints. The resulting solution is not asked to meet the bundle constraints, but a linear programming method is used to update the prices so that solutions generated from the subproblems satisfy the bundle constraints. Iteratively then we are solving a Lagrangian subproblem and a price setting linear program.

The multicommodity problem can be also seen as a capacity allocation problem: all commodities are competing for the available capacity on each arc u_{ij}. The basic idea of resource decomposition methods is to allocate the capacity to the commodities, and then to solve the independent single-commodity flow problems resulting from that allocation. The information gathered when solving the independent single-commodity problems is used to reallocate the capacity to the commodities so the overall system cost is improved.

Partitioning methods exploit the structure of the multicommodity flow problems as special linear programs with embedded network flow problems. Then the question arises of whether we could use linear programming methods that are proven to be efficient (such as the simplex method) to solve the multicommodity flow problem. The partitioning method is a linear programming approach that maintains a linear programming basis that is composed of *spanning trees* of the individual single-commodity flow problems as well as additional arcs that relate the single-commodity problems together to include the bundle constraints.

A.4.1. Optimality Conditions

Multicommodity Arc Flow Complementary Slackness Conditions

The commodity flows y_{ij}^k are optimal in the multicommodity flow problem (A.1) with the $u_{ij}^k = +\infty$ if and only if they are feasible and for some choice of nonnegative arc prices w_{ij} and node potentials $\pi^k(i)$, the reduced costs and arc flows satisfy the following complementary slackness conditions:

$$\text{(a)} \quad w_{ij}\left(\sum_{1\le k\le K} y_{ij}^{k} - u_{ij}\right) = 0 \qquad \forall (i,j)\in A \qquad \text{(A.10)}$$

$$\text{(b)} \quad c_{ij}^{\pi,k} \ge 0 \qquad \forall (i,j)\in A \text{ and } k = 1,2,\ldots,K \qquad \text{(A.11)}$$

$$\text{(c)} \quad c_{ij}^{\pi,k} y_{ij}^{k} = 0 \qquad \forall (i,j)\in A \text{ and } k = 1,2,\ldots,K \qquad \text{(A.12)}$$

where $c_{ij}^{\pi,k}$ is the reduced cost of arc (i,j) with respect to commodity k, ad is defined as follows:

$$c_{ij}^{\pi,k} = c_{ij}^{k} + w_{ij} - \pi^{k}(i) + \pi^{k}(j) \qquad \text{(A.13)}$$

where w_{ij} is the price on arc (i,j), and $\pi^{k}(i)$ is the node potential for each combination of commodity k and node i. Both, w_{ij} and $\pi^{k}(i)$ are the variables in the dual of the problem (A.1).

Multicommodity Path Flow Complementary Slackness Conditions

The commodity path flows $f(P)$ are optimal in the path flow formulation of the multicommodity flow problem (A.6) if and only if for some arc prices w_{ij} and commodity prices σ^{k}, the reduced costs and arc flows satisfy the following complementary slackness conditions:

$$\text{(a)} \quad w_{ij}\left(\sum_{1\le k\le K}\sum_{P\in\mathbf{P^k}} \delta_{ij}(P) - u_{ij}\right) = 0 \qquad \forall (i,j)\in A \qquad \text{(A.14)}$$

$$\text{(b)} \quad c_{P}^{\sigma,w} \ge 0 \qquad \forall (i,j)\in A \text{ and } \forall P\in\mathbf{P^k} \qquad \text{(A.15)}$$

$$\text{(c)} \quad c_{P}^{\sigma,w} f(P) = 0 \qquad \forall (i,j)\in A \text{ and } \forall P\in\mathbf{P^k} \qquad \text{(A.16)}$$

where the reduced cost $c_{P}^{\sigma,w}$ for each path flow $f(P)$ is:

$$c_{P}^{\sigma,w} = c^{k}(P) + \sum_{(i,j)\in P} w_{ij} - \sigma^{k} \qquad \text{(A.17)}$$

B. AMPL: A Modeling Language for Mathematical Programming

B.1. Mathematical Programming

The term *mathematical programming* is used in the common literature about operational research as a way to describe the minimization or maximization of an objective function of many variables, subject to constraints in the variables [36]. One special case in which all the costs and constraints are linear functions of the variables is called a *linear program*, and the process of setting up such a problem and solving it is called *linear programming*. Linear programming is particularly important because a wide variety of problems can be modeled as linear programs, and because the methods to solve it have been well studied, giving birth to fast and reliable methods to solve them. If some nonlinear function of the variables is instead used in the objective or constraints, the problem is referred as a *nonlinear program*. Methods to solve nonlinear programs were developed only recently, after the success of the methods for linear programs. Also, the variables can be asked to take only integral values. Those problems are called *integer programming*.

Practically, solving a problem through mathematical programming requires the following steps:

1. Formulate a model: the system of variables, the objective functions and the system of constraints that represent the general form of the problem to be solved.
2. Collect the data that define a specific problem instance.
3. Generate the specific system of equations from the model and the data instance, which constitute the specific problem to solve.
4. Solve the problem instance by running a program or *solver*.
5. Analyze the results.
6. Refine the model and the data as necessary, and repeat.

If humans could deal with mathematical programs the same way a solver does, the formulation and generation phases of modelling might be relatively straightforward. However, a conversion between the way a human understands a problem (*modeler's form*) and the way a solver represents it (*algorithm's form*) needs to be done. The complexity of this conversion depends on the nature of the problem and mainly on the particular problem instance,

resulting in a time consuming and error-prone procedure.

In the special case of linear programming, the largest part of the solver form of the problem is the *constraint coefficient* matrix, which is typically a very large and sparse matrix (with most entries at 0). Generating the problem instance by hand is practically impossible. A matrix generator is generally used to generate the problem instances from the model and the data. Several programming languages have been designed specifically for writing matrix generators.

Although matrix generators can successfully automate some of the work of translation from modeler's form to algorithm's form, they are difficult to debug and maintain. One way to deal with the difficulties found in designing matrix generators would be to use a *modelling language* for mathematical programming. A modelling language is designed to express the modeler's form in a way that can serve as direct input to *any* solver. In this way, the translation to the algorithm's form can be performed entirely inside the computer system and passed to the solver without the intermediate stage of computer programming.

There is a more than one form to express mathematical programs, so there is more than one kind of modelling language. An *algebraic* modelling language is based on the use of traditional mathematical notation to describe objective and constraints functions. Familiarity to the modeler's mathematical background is one of the major advantages of algebraic modelling. The other important advantage is that it can be easily applied to any kind of linear, nonlinear and integer programming models.

AMPL is an algebraic modelling language for mathematical programming. It was designed and implemented by R. Fourer, D. Gay and B. Kernighan [36] around 1985. AMPL is widely chosen together with another powerful modelling language called GAMS [84], mainly because of the close similarity of its arithmetic expressions to customary algebraic notation, and for the generality and power of its set and subscripting expressions. Both modelling languages have become the standard for the input format used by most available (commercial and non-commercial) solvers. By providing a flexible interface, the user can switch to solve the same problem using any solver implementing the AMPL language without any additional effort. Once the optimal solution found by the solver, it is translated back to modeler's form so that the human user can interpret and analyze them.

B.2. AMPL Basics

The separation of model and data is the key to describing complex linear programs in a concise and understandable fashion. In the model part of the problem, we write a compact

description of the problem in its general form using algebraic notation for the objective and the constraints. The fundamental components to a problem are:

- Sets.
- Parameters.
- Variables.
- Objective.
- Constraints.

The model describes an infinite number of related optimization problems, instantiated by the particular values for data given in each data file. When instantiated, a model becomes a specific problem that can be solved. The advantage of separating model and data in this way is that large instances can be easily dealt with only changing the data file, while the model stays unchanged. In what follows, we present the models defined for the MTDFAP, MPSFAP1 and MPSFAP2 problems defined in Chapter 4 and for the MRPSFAP problem defined in Chapter 7. Data files for some of the models are given as examples.

B.3. Model Files

B.3.1. Model File for MPSFAP1 Problem

```
### Model For Multicommodity Flow Allocation: MPSFAP
#
# Sergio Beker - INFRES-ENST - 03.03.2003
#
# File: MPSFAP.static.mod
#
# Objective: Minimize sum{p in P} path_cost*used_path
# subject to:
# 1) demand constraint: sum{p in Pq} path_flow = demand_q
# 2) capacity constraint: sum{p in Pi} path_flow <= Ci
# 3) path delay constraint: used_path*sum{i in p} link_delay <= max_delay
# 4) path flow constraint: path_flow <= used_path*limit
#
# variables:
# path_flow
# used_path : binary {0,1}

### SETS ###

param N > 0 integer; # number of nodes
param M > 0 integer; # number of links
param K > 0 integer; # maximun number of paths available for a couple.

set nodes := 1..N; # set of nodes

set links := 1..M; # set of links

set path_index := 1..K; # set of paths available for a couple.

set couples := {n1 in nodes, n2 in nodes : n1 <> n2};
# set of couples origin - destination

set path_link_incidence within {(m,n) in couples, k in path_index, i in links};
```

```
# path_link incidence matrix

set paths := setof {(m,n,k,i) in path_link_incidence} (m,n,k);
# list of paths in the path_link incidence matrix

set paths_per_link {i in links} := {(m,n,k,j) in path_link_incidence : i = j};
# list of paths traversing link i

set links_per_path {(m,n,k) in paths} := {j in links : (m,n,k,j) in path_link_incidence};
# list of links used by path (m,n,p)

### PARAMETERS ###

param C {links} >= 0; # link capacity
param e >= 0; # link capacity reduction (to avoid 0 division)

param d {couples} >= 0;
# demand matrix

param path_cost {(m,n,k) in paths} := card{links_per_path[m,n,k]};
# cost per path = number of hops

param B {(m,n,k) in paths} >= 0;
# a limit in the path_flow (to ensure used_path = 0 or 1)

param D {couples} >= 0;
# maximum delay to be admitted for a couple origin, destination

param previous_used_path {paths} binary;

### VARIABLES ###

var used_path {paths} binary; # path is used = 1, path is not used = 0

var changed_path {(m,n,k) in paths} = used_path[m,n,k]*(used_path[m,n,k]-previous_used_path[m,n,k]);
# path has changed = 1, paths has not changed = 0

var path_flow {paths} >= 0; # bp

var link_flow {i in links} = sum{(m,n,k,i) in paths_per_link[i]} path_flow[m,n,k];
# total flow in link i

var link_delay {i in links} = 1/(C[i]-link_flow[i]);
# link delay

var path_delay{(m,n,k) in paths} = sum{i in links_per_path[m,n,k]} link_delay[i];
# delay associated with a path

### OBJECTIVE ###

minimize total_cost : sum {(m,n,k) in paths} (path_cost[m,n,k]*used_path[m,n,k]);

### CONSTRAINTS ###

subject to capacity_constraint {i in links}: link_flow[i] <= (1-e)*C[i];
# Capacity constraint

subject to path_flow_constraint {(m,n,k) in paths} : path_flow[m,n,k] <= B[m,n,k]*used_path[m,n,k];
# constraint to force used_paths to take values when b > 0

subject to demand_constraint {(m,n) in couples}: sum {(m,n,k) in paths} (path_flow[m,n,k]) = d[m,n];
# demand constraint

subject to delay_constraint {(m,n,k) in paths}: path_delay[m,n,k]*used_path[m,n,k] <= D[m,n];
# delay constraint per link

### CALCULATED VARIABLES TO DISPLAY ###

var avg_link_flow = sum{i in links} link_flow[i]/M;

var max_link_flow = max{i in links} link_flow[i];

var active_paths = sum{(m,n,k) in paths} (if path_flow[m,n,k] > 0 then 1 else 0);
# to see the total quantity of active paths;

var active_paths_per_commodity {(m,n) in couples} = sum{(m,n,k) in paths} (if path_flow[m,n,k] > 0 then 1 else 0);
```

```
# active paths per commodity

var active_paths_per_link {i in links} = sum {(m,n,k,i) in path_link_incidence} (if path_flow[m,n,k] > 0 then 1 else 0);

var total_hops = sum{(m,n,k) in paths} (path_cost[m,n,k]*used_path[m,n,k]);

var total_changes = sum {(m,n,k) in paths} changed_path[m,n,k];

var max_path_delay = max{(m,n,k) in paths} path_delay[m,n,k]*used_path[m,n,k];

var avg_path_delay = sum{(m,n,k) in paths} used_path[m,n,k]*path_delay[m,n,k]/active_paths;
# avg delay counting only the used paths;

var total_delay = sum{(m,n,k) in paths} used_path[m,n,k]*path_delay[m,n,k];

var total_cost_of_change = sum{(m,n,k) in paths} changed_path[m,n,k]*path_cost[m,n,k];
# cost of the change part of the objective function (not used in static).
```

B.3.2. Model File for MPSFAP2 Problem

```
### Model For Multicommodity Flow Allocation: MPSFAP
#
# Sergio Beker - INFRES-ENST - 03.03.2003
#
# File: MPSFAP.static.mod
#
# Objective: Minimize sum{p in P} path_cost*used_path
# subject to:
# 1) demand constraint: sum{p in Pq} path_flow = demand_q
# 2) capacity constraint: sum{p in Pi} path_flow <= Ci
# 3) path delay constraint: used_path*sum{i in p} link_delay <= max_delay
# 4) path flow constraint: path_flow <= used_path*limit
#
# variables:
# path_flow
# used_path : binary {0,1}

### SETS ###

param N > 0 integer; # number of nodes
param M > 0 integer; # number of links
param K > 0 integer; # maximun number of paths available for a couple.

set nodes := 1..N; # set of nodes

set links := 1..M; # set of links

set path_index := 1..K; # set of paths available for a couple.

set couples := {n1 in nodes, n2 in nodes : n1 <> n2};
# set of couples origin - destination

set path_link_incidence within {(m,n) in couples, k in path_index, i in links};
# path_link incidence matrix

set paths := setof {(m,n,k,i) in path_link_incidence} (m,n,k);
# list of paths in the path_link incidence matrix

set paths_per_link {i in links} := {(m,n,k,j) in path_link_incidence : i = j};
# list of paths traversing link i

set links_per_path {(m,n,k) in paths} := {j in links : (m,n,k,j) in path_link_incidence};
# list of links used by path (m,n,p)

### PARAMETERS ###

param C {links} >= 0; # link capacity
param e >= 0; # link capacity reduction (to avoid 0 division)

param d {couples} >= 0;
# demand matrix

param path_cost {(m,n,k) in paths} := card{links_per_path[m,n,k]};
# cost per path = number of hops
```

```
param B {(m,n,k) in paths} >= 0;
# a limit in the path_flow (to ensure used_path = 0 or 1)

param D {couples} >= 0;
# maximum delay to be admitted for a couple origin, destination

param previous_used_path {paths} binary;

### VARIABLES ###

var used_path {paths} binary; # path is used = 1, path is not used = 0

var changed_path {(m,n,k) in paths} = used_path[m,n,k]*(used_path[m,n,k]-previous_used_path[m,n,k]);
# path has changed = 1, paths has not changed = 0

var path_flow {paths} >= 0; # bp

var link_flow {i in links} = sum{(m,n,k,i) in paths_per_link[i]} path_flow[m,n,k];
# total flow in link i

var link_delay {i in links} = 1/(C[i]-link_flow[i]);
# link delay

var path_delay{(m,n,k) in paths} = sum{i in links_per_path[m,n,k]} link_delay[i];
# delay associated with a path

### OBJECTIVE ###

minimize total_cost : sum {(m,n,k) in paths} ((path_cost[m,n,k]+path_delay[m,n,k])*used_path[m,n,k]);

### CONSTRAINTS ###

subject to capacity_constraint {i in links}: link_flow[i] <= (1-e)*C[i];
# Capacity constraint

subject to path_flow_constraint {(m,n,k) in paths} : path_flow[m,n,k] <= B[m,n,k]*used_path[m,n,k];
# constraint to force used_paths to take values when b > 0

subject to demand_constraint {(m,n) in couples}: sum {(m,n,k) in paths} (path_flow[m,n,k]) = d[m,n];
# demand constraint

subject to delay_constraint {(m,n,k) in paths}: path_delay[m,n,k]*used_path[m,n,k] <= D[m,n];
# delay constraint per link

### CALCULATED VARIABLES TO DISPLAY ###

var avg_link_flow = sum{i in links} link_flow[i]/M;

var max_link_flow = max{i in links} link_flow[i];

var active_paths = sum{(m,n,k) in paths} (if path_flow[m,n,k] > 0 then 1 else 0);
# to see the total quantity of active paths;

var active_paths_per_commodity {(m,n) in couples} = sum{(m,n,k) in paths} (if path_flow[m,n,k] > 0 then 1 else 0);
# active paths per commodity

var active_paths_per_link {i in links} = sum {(m,n,k,i) in path_link_incidence} (if path_flow[m,n,k] > 0 then 1 else 0);

var total_hops = sum{(m,n,k) in paths} (path_cost[m,n,k]*used_path[m,n,k]);

var total_changes = sum {(m,n,k) in paths} changed_path[m,n,k];

var max_path_delay = max{(m,n,k) in paths} path_delay[m,n,k]*used_path[m,n,k];

var avg_path_delay = sum{(m,n,k) in paths} used_path[m,n,k]*path_delay[m,n,k]/active_paths;
# avg delay counting only the used paths;

var total_delay = sum{(m,n,k) in paths} used_path[m,n,k]*path_delay[m,n,k];

var total_cost_of_change = sum{(m,n,k) in paths} changed_path[m,n,k]*path_cost[m,n,k];
# cost of the change part of the objective function (not used in static).
```

B.3.3. Model File for MTDFAP Problem

```
### Model For Multicommodity Flow Allocation: MTDFAP
#
# Sergio Beker - INFRES-ENST - 03.03.2003
#
# File: MTDFAP.mod
#
# Objective: Minimize sum{p in P} path_delay
# subject to:
# 1) demand constraint: sum{p in Pq} path_flow = demand_q
# 2) capacity constraint: sum{p in Pi} path_flow <= Ci
# 3) path delay constraint: used_path*sum{i in p} link_delay <= max_delay
# 4) path flow constraint: path_flow <= used_path*limit
#
# variables:
# path_flow
# used_path : binary {0,1}

### SETS ###

param N > 0 integer; # number of nodes
param M > 0 integer; # number of links
param K > 0 integer; # maximun number of paths available for a couple.

set nodes := 1..N; # set of nodes

set links := 1..M; # set of links

set path_index := 1..K; # set of paths available for a couple.

set couples := {n1 in nodes, n2 in nodes : n1 <> n2};
# set of couples origin - destination

set path_link_incidence within {(m,n) in couples, k in path_index, i in links};
# path_link incidence matrix

set paths := setof {(m,n,k,i) in path_link_incidence} (m,n,k);
# list of paths in the path_link incidence matrix

set paths_per_link {i in links} := {(m,n,k,j) in path_link_incidence : i = j};
# list of paths traversing link i

set links_per_path {(m,n,k) in paths} := {j in links : (m,n,k,j) in path_link_incidence};
# list of links used by path (m,n,p)

### PARAMETERS ###

param C {links} >= 0; # link capacity
param e >= 0; # link capacity reduction (to avoid 0 division)

param d {couples} >= 0;
# demand matrix

param path_cost {(m,n,k) in paths} := card{links_per_path[m,n,k]};
# cost per path = number of hops

param B {(m,n,k) in paths} >= 0;
# a limit in the path_flow (to ensure used_path = 0 or 1)

param D {couples} >= 0;
# maximum delay to be admitted for a couple origin, destination

param previous_used_path {paths} binary;

### VARIABLES ###

var used_path {paths} binary; # path is used = 1, path is not used = 0

var changed_path {(m,n,k) in paths} = used_path[m,n,k]*(used_path[m,n,k]-previous_used_path[m,n,k]);
# path has changed = 1, paths has not changed = 0

var path_flow {paths} >= 0; # bp

var link_flow {i in links} = sum{(m,n,k,i) in paths_per_link[i]} path_flow[m,n,k];
# total flow in link i
```

```
var link_delay {i in links} = 1/(C[i]-link_flow[i]);
# link delay

var path_delay{(m,n,k) in paths} = sum{i in links_per_path[m,n,k]} link_delay[i];
# delay associated with a path

### OBJECTIVE ###

minimize total_cost : sum {(m,n,k) in paths} (path_delay[m,n,k]*used_path[m,n,k]);

### CONSTRAINTS ###

subject to capacity_constraint {i in links}: link_flow[i] <= (1-e)*C[i];
# Capacity constraint

subject to path_flow_constraint {(m,n,k) in paths} : path_flow[m,n,k] <= B[m,n,k]*used_path[m,n,k];
# constraint to force used_paths to take values when b > 0

subject to demand_constraint {(m,n) in couples}: sum {(m,n,k) in paths} (path_flow[m,n,k]) = d[m,n];
# demand constraint

subject to delay_constraint {(m,n,k) in paths}: path_delay[m,n,k]*used_path[m,n,k] <= D[m,n];
# delay constraint per link

### CALCULATED VARIABLES TO DISPLAY ###

var avg_link_flow = sum{i in links} link_flow[i]/M;

var max_link_flow = max{i in links} link_flow[i];

var active_paths = sum{(m,n,k) in paths} (if path_flow[m,n,k] > 0 then 1 else 0);
# to see the total quantity of active paths;

var active_paths_per_commodity {(m,n) in couples} = sum{(m,n,k) in paths} (if path_flow[m,n,k] > 0 then 1 else 0);
# active paths per commodity

var active_paths_per_link {i in links} = sum {(m,n,k,i) in path_link_incidence} (if path_flow[m,n,k] > 0 then 1 else 0);

var total_hops = sum{(m,n,k) in paths} (path_cost[m,n,k]*used_path[m,n,k]);

var total_changes = sum {(m,n,k) in paths} changed_path[m,n,k];

var max_path_delay = max{(m,n,k) in paths} path_delay[m,n,k]*used_path[m,n,k];

var avg_path_delay = sum{(m,n,k) in paths} used_path[m,n,k]*path_delay[m,n,k]/active_paths;
# avg delay counting only the used paths;

var total_delay = sum{(m,n,k) in paths} used_path[m,n,k]*path_delay[m,n,k];

var total_cost_of_change = sum{(m,n,k) in paths} changed_path[m,n,k]*path_cost[m,n,k];
# cost of the change part of the objective function (not used in static).
```

B.3.4. Model File for MRPSFAP Problem

```
### Model For Multicommodity Flow Allocation: MINLP1a - Dynamic Formulation
#
# Sergio Beker - INFRES-ENST - 03.03.2003
#
# File: MINLP1a.dynamic.mod
#
# Objective: Minimize sum{p in P} path_cost*used_path + sum {p in P} path_cost*changed_path
# subject to:
# 1) demand constraint: sum{p in Pq} path_flow = demand_q
# 2) capacity constraint: sum{p in Pi} path_flow <= Ci
# 3) path delay constraint: used_path*sum{i in p} link_delay <= max_delay
# 4) path flow constraint: path_flow <= used_path*limit
# 5) change constraint: (used_path(t+1)-used_path(t))*used_path(t+1) <= changed_path(t+1)
#
# variables:
# path_flow
# used_path : binary {0,1}
# changed_path : binary {0,1}
```

```
### SETS ###

param N > 0 integer; # number of nodes
param M > 0 integer; # number of links
param K > 0 integer; # maximun number of paths available for a couple.

set nodes := 1..N; # set of nodes

set links := 1..M; # set of links

set path_index := 1..K; # set of paths available for a couple.

set couples := {n1 in nodes, n2 in nodes : n1 <> n2};
# set of couples origin - destination

set path_link_incidence within {(m,n) in couples, k in path_index, i in links};
# path_link incidence matrix

set paths := setof {(m,n,k,i) in path_link_incidence} (m,n,k);
# list of paths in the path_link incidence matrix

set paths_per_link {i in links} := {(m,n,k,j) in path_link_incidence : i = j};
# list of paths traversing link i

set links_per_path {(m,n,k) in paths} := {j in links : (m,n,k,j) in path_link_incidence};
# list of links used by path (m,n,p)

### PARAMETERS ###

param C {links} >= 0; # link capacity
param e >= 0; # link capacity reduction (to avoid 0 division)

param d {couples} >= 0;
# demand matrix

param path_cost {(m,n,k) in paths} := card{links_per_path[m,n,k]};
# cost per path = number of hops

param B {(m,n,k) in paths} >= 0;
# a limit in the path_flow (to ensure used_path = 0 or 1)

param D {couples} >= 0;
# maximum delay to be admitted for a couple origin, destination

param previous_used_path {paths} binary;

### VARIABLES ###

var used_path {paths} binary; # path is used = 1, path is not used = 0

var changed_path {paths} binary;
# path has changed = 1, paths has not changed = 0

var path_flow {paths} >= 0; # bp

var link_flow {i in links} = sum{(m,n,k,i) in paths_per_link[i]} path_flow[m,n,k];
# total flow in link i

var link_delay {i in links} = 1/(C[i]-link_flow[i]);
# link delay

var path_delay{(m,n,k) in paths} = sum{i in links_per_path[m,n,k]} link_delay[i];
# delay associated with a path

### OBJECTIVE ###

minimize total_cost : sum {(m,n,k) in paths} (path_cost[m,n,k]*used_path[m,n,k]) + sum {(m,n,k) in paths} (path_cost[m,n,k]*changed_path[m,n,k]);

### CONSTRAINTS ###

subject to capacity_constraint {i in links}: link_flow[i] <= (1-e)*C[i];
# Capacity constraint

subject to path_flow_constraint {(m,n,k) in paths} : path_flow[m,n,k] <= B[m,n,k]*used_path[m,n,k];
# constraint to force used_paths to take values when b > 0
```

```
subject to demand_constraint {(m,n) in couples}: sum {(m,n,k) in paths} (path_flow[m,n,k]) = d[m,n];
# demand constraint

subject to delay_constraint {(m,n,k) in paths}: path_delay[m,n,k]*used_path[m,n,k] <= D[m,n];
# delay constraint per link

subject to path_change {(m,n,k) in paths} : (used_path[m,n,k]-previous_used_path[m,n,k])*used_path[m,n,k] <= changed_path[m,n,k];

### CALCULATED VARIABLES TO DISPLAY ###

var avg_link_flow = sum{i in links} link_flow[i]/M;

var max_link_flow = max{i in links} link_flow[i];

var active_paths = sum{(m,n,k) in paths} (if path_flow[m,n,k] > 0 then 1 else 0);
# to see the total quantity of active paths;

var active_paths_per_commodity {(m,n) in couples} = sum{(m,n,k) in paths} (if path_flow[m,n,k] > 0 then 1 else 0);
# active paths per commodity

var active_paths_per_link {i in links} = sum {(m,n,k,i) in path_link_incidence} (if path_flow[m,n,k] > 0 then 1 else 0);

var total_hops = sum{(m,n,k) in paths} (path_cost[m,n,k]*used_path[m,n,k]);

var total_changes = sum {(m,n,k) in paths} changed_path[m,n,k];

var max_path_delay = max{(m,n,k) in paths} path_delay[m,n,k]*used_path[m,n,k];

var avg_path_delay = sum{(m,n,k) in paths} used_path[m,n,k]*path_delay[m,n,k]/active_paths;
# avg delay counting only the used paths;

var total_cost_of_change = sum{(m,n,k) in paths} path_cost[m,n,k]*changed_path[m,n,k];
```

B.4. Data Files

B.4.1. Data File for MPSFAP and MTDFAP Problems

```
# NET2.dat
# Sergio Beker - ENST - INFRES: 27/01/03
#
# Multicommodity Flow Problem
# nodes = 4
# links = 10
# demands = 1.
# paths = 3 between nodes 1 and 4

data;

param e := 0.0001;
param D default 0.03;
param B default 10000;

param N := 4;
param M := 10;
param K := 5;

param C := 1  2500.0
2  2500.0
3  2500.0
4  2500.0
5  2500.0
6 2500.0
7 2500.0
8 2500.0
9 2500.0
10 2500.0
;

set path_link_incidence :=
(1,2,1,*) 1
(1,2,2,*) 9   4
(1,2,3,*) 8   6   4
```

```
(1,3,1,*) 8
(1,3,2,*) 9   5
(1,3,3,*) 1   3   5

(1,4,1,*) 9
(1,4,2,*) 1   3
(1,4,3,*) 8   6

(2,1,1,*) 2
(2,1,2,*) 3   10
(2,1,3,*) 3   5   7

(2,3,1,*) 2   8
(2,3,2,*) 3   5
(2,3,3,*) 2   9   5
(2,3,4,*) 3   10   8

(2,4,1,*) 3
(2,4,2,*) 2   9
(2,4,3,*) 2   8   6

(3,1,1,*) 7
(3,1,2,*) 6   10
(3,1,3,*) 6   4   2

(3,2,1,*) 6   4
(3,2,2,*) 7   1
(3,2,3,*) 6   10   1
(3,2,4,*) 7   9   4

(3,4,1,*) 6
(3,4,2,*) 7   9
(3,4,3,*) 7   1   3

(4,1,1,*) 10
(4,1,2,*) 4   2
(4,1,3,*) 5   7

(4,2,1,*) 4
(4,2,2,*) 10   1
(4,2,3,*) 5   7   1

(4,3,1,*) 5
(4,3,2,*) 10   8
(4,3,3,*) 4   2   8
;

param previous_used_path default 0;

param d default 0.0 :
1 2 3 4 :=
1 . 346 729 1524
2 37 . 1186 1584
3 1477 1082 . 1834
4 1788 706 19 . ;
```

B.4.2. Data File for MRPSFAP Problem

```
# NET2.dat
# Sergio Beker - ENST - INFRES: 27/01/03
#
# Multicommodity Flow Problem
# nodes = 4
# links = 10
# demands = 1.
# paths = 3 between nodes 1 and 4

data;

param e := 0.0001;
param D default 0.03;
param B default 10000;
```

```
param N := 4;
param M := 10;
param K := 5;

param C := 1  2500.0
2  2500.0
3  2500.0
4  2500.0
5  2500.0
6 2500.0
7 2500.0
8 2500.0
9 2500.0
10 2500.0
;

set path_link_incidence :=
(1,2,1,*) 1
(1,2,2,*) 9   4
(1,2,3,*) 8   6   4

(1,3,1,*) 8
(1,3,2,*) 9   5
(1,3,3,*) 1   3   5

(1,4,1,*) 9
(1,4,2,*) 1   3
(1,4,3,*) 8   6

(2,1,1,*) 2
(2,1,2,*) 3   10
(2,1,3,*) 3   5   7

(2,3,1,*) 2   8
(2,3,2,*) 3   5
(2,3,3,*) 2   9   5
(2,3,4,*) 3   10   8

(2,4,1,*) 3
(2,4,2,*) 2   9
(2,4,3,*) 2   8   6

(3,1,1,*) 7
(3,1,2,*) 6   10
(3,1,3,*) 6   4   2

(3,2,1,*) 6   4
(3,2,2,*) 7   1
(3,2,3,*) 6   10   1
(3,2,4,*) 7   9   4

(3,4,1,*) 6
(3,4,2,*) 7   9
(3,4,3,*) 7   1   3

(4,1,1,*) 10
(4,1,2,*) 4   2
(4,1,3,*) 5   7

(4,2,1,*) 4
(4,2,2,*) 10   1
(4,2,3,*) 5   7   1

(4,3,1,*) 5
(4,3,2,*) 10   8
(4,3,3,*) 4   2   8
;

param previous_used_path [1,*,*] (tr)
:   2   3   4    :=
1   1   1   1
2   0   0   0
3   0   0   0

 [2,*,*] (tr)
:   1   3   4    :=
```

```
1   1   1   1
2   0   0   0
3   0   0   0
4   .   0   .

 [3,*,*] (tr)
:   1   2   4    :=
1   1   1   1
2   0   1   0
3   0   0   0
4   .   0   .

 [4,*,*] (tr)
:   1   2   3    :=
1   1   1   1
2   0   0   0
3   0   0   0
;

param d default 0.0 :
1 2 3 4 :=
1 . 346 1208 298
2 37 . 1186 1584
3 1477 1082 . 1834
4 1788 706 19 . ;
```

Bibliography

[1] R. Yavatkar, D. Pendarakis, R. Guerin, *A Framework for Policy-based Admission Control*, RFC 2573, January 2000, IETF. 3.1.2

[2] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry, *The COPS (Common Open Policy Service) Protocol*, RFC 2748, January 2000, IETF. 3.1.2

[3] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, A. Smith, *COPS Usage for Policy Provisioning (COPS-PR)*, RFC 3084, March 2001, IETF. 3.1.2

[4] R. Sahita, S. Hahn, K. Chan, K. McCloghrie, *Framework Policy Information Base*, RFC 3318, March 2003, IETF. 3.1.2

[5] S . Herzog, J. Boyle, R. Cohen, D. Durham, R. Rajan, A. Sastry, *COPS usage for RSVP*, RFC 2749, January 2000, IETF. 3.1.2

[6] D. Awduche, *MPLS Traffic Engineering in IP Networks*, IEEE Communications Magazine, December 1999, IEEE. 3, 3.1.1

[7] D. Awduche et al., *Requirements for Traffic Engineering Over MPLS*, RFC 2702, September 1999, IETF.

[8] R. Callon et al., *A Framework for Multiprotocol Label Switching*, Internet Draft, September 1999, IETF. 3.1.3

[9] E. Rosen, A. Viswanathan, R. Callon, *Multiprotocol Label Switching Architecture*, RFC3031, January 2001, IETF. 3.1.3

[10] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, *RSVP-TE: RSVP Extensions for MPLS Tunnels*, RFC3209, December 2001, IETF. 3.3.1, 4.1, 7.3

[11] F. Le Faucheur, B. Davie, S. Davari, P. Vaanen, R. Krishnan, P. Cheval, J. Heinanen, *Multiprotocol Label Switching (MPLS) Support for Differentiated Services*, RFC3270, May 2002, IETF. 3.1.3, 4.7

[12] B. Jamoussi et al., *Constraint-Based LSP Setup using LDP*, RFC 3212, January 2002, IETF. 3.3.1, 4.1

[13] D. Awduche et al., *RSVP-TE: Extensions to RSVP for LSP Tunnels*, RFC 3209, December 2001, IETF.

[14] R.Ahuja, T.Magnanti, J.Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall Inc., 1993, p.788-801. 4.6, A.1, A.3, A.4

[15] M.Kodialam, T.Lakshman, *Minimum Interference Routing with Applications to MPLS Traffic Engineering*, INFOCOM 2000, Proceedings IEEE, Volume 2. p.884-893. 3.3.1, 4.8

[16] S.Beker, D.Kofman, N.Puech, *Off-Line Reduced Complexity Layout Design for MPLS Networks*, Proceedings of IPOM 2003, p.99-105, IEEE. 4.3, 7.1

[17] S.Beker, D.Kofman, N.Puech, *Off-Line MPLS Layout Design and Reconfiguration : Reducing Complexity Under Dynamic Traffic Conditions* Proceedings of INOC 2003, p.61-66, ITC/INFORMS.

[18] J. Czyzyk, M. Mesnier, and J. Moré, *The Neos Server*, IEEE Journal on Computational Science and Engineering, 5, pages 68-75. 4.6.4

[19] R. Fletcher, S. Leyffer, *Numerical Experience with Lower Bounds for MIQP Branch and Bound*, SIAM Journal of Optimization, 8(2):p.604-616, 1998. 4.6.4

[20] Roberts J.W., *Self-Similar Network Traffic and Performance Evaluation, chapter Engineering for Quality of Service*, Wiley-Interscience, 2000, pages 401-420. 4.2

[21] Cabinet Arcome, *Etude Technique, Economique et Réglamentaire de l'Evolution vers les Réseaux de Nouvelle Génération*, Autoritéde Règulation des Télécommunications, Septembre 2000. 2.3.3, 2.3.4

[22] E. Altman, T. Başar, T. Jiménez, N. Shimkin, *Competitive Routing in Networks with Polynomial Costs*, INFOCOM 2000, Proceedings IEEE, Volume 3, p.1586-1593. 4.1, 4.3, 4.8

[23] L. Georgiadis, K. Floros, P. Georgatsos, S. Sartzetakis, *Lexicographically Optimal Balanced Networks*, INFOCOM 2001, Proceedings IEEE, Volume 2, p.689-698. 4.1, 4.3, 4.8

[24] H. Hsu, F. Yeang-Sung Lin, *Near-Optimal Constrained Routing in Virtual Circuit Networks*, INFOCOM 2001, Proceedings IEEE, Volume 2, p.750-756. 4.1, 4.3, 4.8

[25] D. Banerjee, B. Mukherjee, *Wavelength-Routed Optical Networks: Linear Formulation, Resource Budgeting Tradeoffs, and a Reconfiguration Study*, Networking, IEEE/ACM Transactions on, Volume: 8 Issue: 5 , Oct 2000, p.598-607. 3.3.1, 4.1, 4.6.1, 7.1, 7.6.1

[26] G. Banerjee, D. Sidhu, *Path Computation for Traffic Engineering in MPLS Networks*, Proceedings of IEEE ICN 2001.

[27] A. Feldman, A.C. Gibert, P. Huang, W. Willinger, *Dynamics of IP Traffic: A Study of the Role of Variability and the Impact of Control*, SIGCOM 1999. 4.2

[28] M. Nabe, M. Murata, H. Miyahara, *Analysis and Modeling of Worldwide Web Traffic for Capacity Dimensioning of Internet Access Lines* Performance Evaluation, (34):249-271, 1998. 4.2

[29] K. Thompson, G.J. Miller, R. Wilder, *Wide-Area Internet Traffic Patterns and Characterization*, IEEE Network, Nov/Dec. 1997, pages: 10-23. 4.2

[30] L. Kleinrock, *Queueing systems*, Vol. 1 : Theory, John Wiley & Sons, New York, USA. 1975. 4.3

[31] L. Kleinrock, *Queueing systems*, Vol. 2 : Computer Applications, John Wiley & Sons, New York, USA. 1976. 5.1, 8.5

[32] T. Wang, *Global Optimization for Constrained Non Linear Programming*, PhD. Thesis, University of Illinois at Urbana-Champaign, 2001. 4.6

[33] Michael R. Garey, David. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freemand and Co. New York, 1979. 4.6

[34] George L. Nemhouser, Laurence A. Walsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, New York, 1988. 4.6

[35] M. Ajmone Marsan, A. Grosso, E. Leonardi, M. Mellia, A. Nucci, *Design of Logical Topologies in Wavelength-Routed IP Networks*, Photonic Network Communications, December 2002, Vol.4, No.3/4, pp.423-442, ISSN: 1387-974x. 4.6

[36] Robert Fourer, David M. Gay, Brian W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, Second Edition, Thompson, Brooks/Cole, USA, 2003. 4.6.4, B.1, B.1

[37] W. Gropp and J. Moré, *Optimization Environments and the NEOS Server*, Approximation Theory and Optimization, M. D. Buhmann and A. Iserles, eds., pages 167-182, 1997, Cambridge University Press. 4.6.4

[38] E. Dolan, *The NEOS Server 4.0 Administrative Guide*, Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, May 2001. 4.6.4

[39] R. Fletcher, S. Leyffer, *User Manual for Filter SQP*, Numerical Analysis Report, NA/181, Dundee University, April, 1998. 4.6.4

[40] W. Ben-Ameur, B. Liau, N. Michel, *Routing Strategies for IP Networks*, 2/3 2001, Telektronik Magazine, p. 145-158. 3.1.3, 4.8, 6.2

[41] S. C. Erbas, R. Mathar:, *An Off-line Traffic Engineering Model for MPLS Networks*, Proceddings of IEEE 27th Annual IEEE Conference on Local Computer Networks (27th LCN), pp. 166-174, Tampa, Florida, November 2002.

[42] M. Pióro, A. Myslek, *Topological Design of MPLS Networks*, IEEE GLOBECOM 2001, San Antonio, Texas, 2001. 4.1

[43] D. Bienstock, S. Chopra, O. Guenluek, C.Y. Tsai, *Minimum Cost Capacity Installation for Multicommodity Network Flows*, Mathematical Programming Journal, (81), p.177-199, 1998. 4.1

[44] N. Geary, A. Antonopoulos, E. Drakopoulos, J. O'Reilly, *Analysis Of Optimisation Issues In Multi-Period DWDM Network Planning*, IEEE INFOCOM 2001, Anchorage, Alaska, p.152-158. 4.1

[45] B. Fortz, M. Thorup, *Internet Traffic Engineering by Optimizing OSPF Weights*, IEEE INFOCOM, p.519-528, 2000. 3.1.2, 3.1.3, 4.8, 7.1

[46] F. Ergun, R. Sinha, L. Zhang, *QoS Routing with PerformanceDependent Costs*, IEEE INFOCOM, 2000. 4.8

[47] A. Orda, A. Sprintson, *QoS Routing: The Precomputation Perspective*, IEEE INFOCOM, p.128-136, 2000. 4.8

[48] T. Korkmaz, M. Krunz, *Multi-Constrained Optimal Path Selection*, IEEE INFOCOM, p.834-843, 2001. 4.8

[49] A. Jüttner, B. Szviatovszki, I. Mécs, Z. Rajkó, *Lagrange Relaxation Based Method for the QoS Routing Problem*, IEEE INFOCOM, 2001. 4.8

[50] A. Goel, K.G. Ramakrishnan, D. Kataria, D. Logothetis, *Efficient Computation of Delay-sensitive Routes from One Source to All Destinations*, IEE INFOCOM, p.854-858, 2001. 4.8

[51] O.K. Gupta, A. Ravindran, *Branch and Bound Experiments in Convex Non-Linear Integer Programming*, Management Science, 31(12), p.1533-1546, 1985. 5.1

[52] I. Quesada, I.E. Grossmann, *An LP/NLP Based Branch and Bound Algorithm for Convex MINLP Optimization Problems*, Computers Chemical Engineering, 16(10/11), p.937-947, 1992. 5.1

[53] A.M. Geoffrion, *A Generalized Benders Decomposition*, Journal on Optimization Theory and Applications, 10(4), p.237-260, 1972. 5.1

[54] M.A. Duran, I.E. Grossmann, *An Outer-Approximation Algorithm for a Class of Mixed-Integer Non-Linear Programming*, Mathematical Pogramming, 36, p.307-339, 1986. 5.1

[55] R. Fletcher, S. Leyffer, *Solving Mixed Integer Programs by Outer Approximation*, Mathematical Pogramming, 66, p.327-349, 1994. 5.1

[56] S. Leyffer, *Integrating SQP and branch-and-bound for Mixed Integer Nonlinear Programming*, Computational Optimization and Applications, 18, p.295-309, 2001. 5.1

[57] T. Westerlund, F. Petersson, *A Cutting Plane Method for Solving Convex MINLP Problems*, Computers Chemical Engineering, 19, p.131-136, 1995. 5.1

[58] S. Kirkpatrick, C.D. Gelatt Jr.,M.P. Vecchi, *Optimization by Simulated Annealing*, Science Journal, number 4598, 1983. 5.2

[59] C. Gazen, C. Ersoy, *Genetic algorithms for designing multihop lightwave network topologies*, Artificial Intelligence in Engineering, (3), p.211-221, 1999. 5.2

[60] F. Glover, M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1997, Boston, MA. 5.2

[61] N. Puech, J. Kuri, M. Gagnaire, *Models for the Logical Topology Design Problem*, Proceedings of the 2nd IFIP-TC6 Networking Conference, May, 2002, Springer-Verlag. 5.2

[62] S. Beker, N. Puech, V. Friderikos, *A Tabu Search Heuristic for the Off-Line MPLS Reduced Complexity Layout Design Problem*, IFIP-TC6 Networking Conference 2004, LNCS Springer Verlag Networking 2004 (to be published) May 9-14, Athens. 5.3

[63] *Réseau National de Reherche en Téléommunications*, 2000-2004, Ministère de l'Economie, des Finances et de l'Industrie, France, www.vthd.org. 1, 6.1

[64] D. Bertsekas, R. Gallager, *Data Networks*, Prentice-Hall International Editions, UK, 1987. 3.1.2, 5.1, 5.4.1, 5.4.1, 5.5, 8.5

[65] J.E. Burns, T.J. Ott, Johan M. de Kock, A.E. Krzesinski, *Path Selection and Bandwidth Allocation in MPLS Networks: a Non-Linear Programming Approach*, Proceedings of SPIE, ITCom (4523), 2001. 5.1, 5.5, 1, 5.5

[66] A. Kershenbaum, *Telecommunication Design Algorithms*, McGraw-Hill, 1993.

[67] L. Fratta, M. Gerla, L. Kleinrock, *The Flow Deviation Method: An Approach to Store-and-Forward Communication Network Design*, Networks, (3), p.97-133, 1973. 5.4, 5.4.1

[68] M. Lucas, D. Wrege, B. Dempsey, A. Weaver, *Statistical Characterization of Wide-Area IP Traffic*, Sixth International Conference on Computer Communications and Networks (IC3N'97), Las Vegas, NV, September 1997. (document), 7.1, 7.2, 7.1

[69] T. Bonald, S. Oueslati-Boulahia, J.W. Roberts, *IP traffic and QoS control: the need for a flow-aware architecture*, World Telecommunications Congress, September 2002 Paris, France. (document), 7.1, 7.2, 7.2

[70] A. Genąta, B. Mukherjee, *Virtual-Topology Adaptation for WDM Lesh Networks Under Dynamic Traffic*, INFOCOM 2002, Proceedings IEEE, (1) p.48-56, 2002. 7.1

[71] B. Ramamurthy, A. Ramakrishnan, *Virtual Topology Reconfiguration of Wavelength-Routed Optical WDM Networks*, GLOBECOM 2000, Proceedings IEEE, (2) p.1269-1275, 2000. 7.1

[72] F. Ricciato, S. Salsano, A. Belmonte, M. Listanti, *Off-Line Configuration of a MPLS over WDM Network under Time-Varying Offered Traffic*, INFOCOM 2002, Proceedings IEEE, (1) p.57-65, 2002. 7.1

[73] W. Ben-Ameur, H. Kerivin, *Routing of uncertain demands*, submitted to Journal on Operations Research, 2001. 7.1

[74] J.F.P. Labourdette, G.W.Hart, A.S.Acampora, *Branch-Exchange Sequences for Reconfiguration of Lightwave Networks*, IEEE Transactions on Communications, 42(10) p.2822-2832, October 1994. 7.1

[75] B. Fortz, J. Rexford, M. Thorup, *Traffic Engineering with Traditional IP Routing Protocols*, IEEE Communications Magazine, 2002. 3.1.3, 7.1

[76] R. Casellas, *Tesis*, ENST, 2002. 3.1.2

[77] K. Ramakrishnan, S. Floyd, D. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, RFC 3178, IETF, Septemnber 2001. 3.1.2

[78] K. Ishiguro, T. Takada, *Traffic Engineering Extensions to OSPF version 3*, Draft (draft-ietf-ospf-ospfv3-traffic-01.txt), IETF. August 2003. 3.1.3

[79] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss *An Architecture for Differentiated Services*, RFC 2475, IETF, August 1998. 3.1.3

[80] J. Wroclawski, *The Use of RSVP with IETF Integrated Services*, RFC 2210, IETF, September 1997. 3.1.3

[81] D. Grossman, *New Terminology and Clarifications for Diffserv*, RFC 3260, IETF, April 2002. 3.1.3

[82] M. Grossglauser, J. Rexford, *Passive Traffic Measurement for IP Operations*, The Internet as a Large-Scale Complex System (Chapter), Oxford University Press, to appear 2004. 3.2.1

[83] V. Paxson, G. Almes, J. Mahdavi, M. Mathis, *Framework for IP Performance Metrics*, RFC 2330, IETF, May 1998. 3.2.1

[84] E. Castillo, A.J. Conejo, P. Pedregal, R. Garcia, N. Alguacil, *Building and Solving Mathematical Programming Models in Engineering and Science*, ISBN: 0-471-15043-6, Hardcover, 568 pages, October 2001. 4.6.4, B.1

[85] A. Feldman, A. Greenberg, C. Lund, N. Reingold, J. Rexford, F. True, *Deriving Traffic Demands for Operational IP Networks: Methododoly and Experience*, Proceedings of ACM SIGCOM 2000, , 2000. 4.2

[86] B. Krithikaivasan, K. Deka, D. Medhi, *Adaptive Bandwidth Provisioning Envelope Based on Discrete Temporal Network Measurements*, Proceedings of IEEE INFOCOM 2004, Hong Kong, March 2004. 4.2

[87] K. Papagiannaki, N. Taft, Z.L. Zhang, C. Diot, *Long-term Forecasting of Internet Backbone Traffic: Observations and Initial Models*, Proceedings of IEEE INFOCOM 2003, San Francisco, CA, March 2003. 4.2

[88] S. Srivastava, B. Krithikaivasan, D. Medhi, M. Pióro, *Traffic Engineering in the Presence of Tunneling and Diversity Constraints: Formulation and Lagrangian Decomposition Approach*, Proceedings of 18^{th} International Teletraffic Congress (ITC18), pages 461-470, September 2003. 4.8

[89] B. Yaged, *Minimum Cost Routing for Dyanamic Network Models*, Networks, 3:193-224, 1973. 7.1

[90] N. Zadeh, *On Building Minimum Cost Communication Networks Over Time*, Networks, 4:19-34, 1974. 7.1