Conception de réseaux de télécommunications : optimisation et expérimentations

Jean-François Lalande

UNIVERSITÉ de NICE-SOPHIA ANTIPOLIS – UFR SCIENCES

École Doctorale STIC

THÈSE

pour obtenir le titre de

Docteur en SCIENCES

de l'Université de Nice-Sophia Antipolis

Discipline : **INFORMATIQUE**

présentée et soutenue par

Jean-François LALANDE

Conception de réseaux de télécommunications : optimisation et expérimentations

Thèse dirigée par **Jean-Claude BERMOND** et **Michel SYSKA**

et préparée à l'INRIA Sophia, projet MASCOTTE

soutenue le **10 décembre 2004**

Jury :

Examinateurs	M. **Jean-Marc**	**FÉDOU**	Professeur
	M. **Philippe**	**MAHEY**	Professeur
	M. **Michel**	**SYSKA**	Maître de Conférences
Directeur	M. **Jean-Claude**	**BERMOND**	Directeur de Recherche
Rapporteurs	M. **Abdelmadjid**	**BOUABDALLAH**	Professeur
	M. **Ricardo**	**CORRÊA**	Professeur
	M. **Samir**	**TOHMÉ**	Professeur

À Anne et Jacques.

Remerciements

Je souhaite remercier en premier lieu Jean-Claude Bermond, mon directeur de thèse, qui m'a donné la chance de travailler dans l'équipe MASCOTTE. Je lui suis d'autant plus redevable qu'il a su me donner, ainsi qu'à toute l'équipe, l'élan et la motivation nécessaires pour que j'aborde toutes les difficultés rencontrées avec un grand enthousiasme et l'assurance de toujours avoir son soutien.

Vient ensuite le directeur "officieux" de cette thèse : Michel Syska. Je tiens à lui témoigner ma gratitude pour le temps, les efforts et les conseils qui m'ont permis de savoir où j'allais, lorsque j'étais perdu : je lui suis reconnaissant d'avoir réussi à diriger mes travaux tout en me laissant libre d'explorer mes pistes à ma guise. Enfin, je tiens à le remercier pour avoir su m'apprendre quelles sont les vraies valeurs d'une recherche académique scientifique de qualité.

Tout au long de ma thèse, j'ai eu le privilège de travailler avec de nombreuses personnes en profitant de l'avantage d'appartenir à une équipe composée de plusieurs instituts de recherche, l'INRIA, l'I3S regroupant le CNRS et l'UNSA et participant à une collaboration avec FRANCE TÉLÉCOM.

Sans vouloir faire de distinguo, je remercie donc chaque chercheur de l'équipe MASCOTTE, à savoir : Bruno Beauquier, Jean-Claude Bermond, Sébastien Choplin, Michel Cosnard, David Coudert, Olivier Dalle, Ephie Deriche, Afonso Ferreira, Jérôme Galtier, Frédéric Havet, Gurvan Huiban, Aubin Jarry, Ralf Klasing, Alexandre Laugier, Nelson Morales, Philippe Mussi, Fabrice Peix, Stéphane Pérennes, Séverine Petat, Hervé Rivano, Jean-Sébastien Sereni, Michel Syska, Corinne Touati, Yann Verhoeven, et Marie-Émilie Voge.

Je souhaite aussi remercier les personnes qui furent de passage dans MASCOTTE et qui contribuèrent de manière décisive à mon travail : Bruno Bongiovanni, David Sagnol et Christian Vallebella.

Je tiens aussi à remercier mes collègues de l'équipe MAESTRO, non pas seulement parce que nous travaillons à leurs côtés, mais surtout parce qu'ils m'ont donné la chance d'aborder un sujet de leur domaine en profitant de leur expertise. Je tiens notamment à remercier nommément Sara Alouf et Eitan Altman.

Je souhaite aussi adresser mes remerciements aux chercheurs du LIMOS à Clermont-Ferrand qui m'ont donné envie de devenir docteur, en me passionnant pour la recherche : merci à Bruno Bachelet, Michel Gourgand et Philippe Mahey avec qui j'espère pouvoir encore continuer à travailler.

Ce manuscrit à été rapporté par trois chercheurs à qui j'adresse mes remerciements. Ils ont pris le temps de relire consciencieusement mon manuscrit et d'apporter leurs critiques et leurs conseils malgré leur emploi du temps chargé de cette fin d'année. Merci à Abdelmadjid Bouabdallah (Université de Technologie de Compiègne), Ricardo Cordeiro Corrêa (Universidade Federal do Ceará, Brésil) et à Samir Tohmé (Université de Versailles).

Le jury de ma soutenance m'a fait l'honneur d'apprécier la présentation de ma thèse et d'apporter leurs suggestions sur la manière et le fond de mes travaux de recherche. Je remercie Jean-Marc Fédou (Université de Nice-Sophia Antipolis), Philippe Mahey (Université Blaise Pascal Clermont II), Michel Syska (Université de Nice-Sophia Antipolis), Jean-Claude Bermond (INRIA, I3S) et Abdelmadjid Bouabdallah (Université de Technologie de Compiègne).

Enfin, je n'oublie pas de signifier mon amitié à mes amis et à ma famille, aux Gros-Nazes, à mes parents, mon frère et mes sœurs.

Je dois aussi une fière chandelle à Sylvie qui a supporté la partie *off* de cette thèse...

Table des matières

Remerciements i

Glossaire des sigles ix

Table des figures xi

Plan de la thèse 1

Première partie . Algorithmes pour la conception de réseaux de cœur 3

Introduction 5

Chapitre 1. Conception de réseaux de cœur 7
1.1. Caractéristiques des réseaux WDM 8
1.1.1. Le multiplexage en longueurs d'onde 8
1.1.2. Niveau des conteneurs 8
1.1.3. Brassage des conteneurs 9
1.1.4. Conversion de longueurs d'onde dans un réseau tout optique 10
1.1.5. Représentation des fonctionnalités des brasseurs 10
1.2. Modèle pour les réseaux WDM 11
1.2.1. Matrice de trafic 11
1.2.2. Multiplicité en longueurs d'onde 12
1.3. Les problèmes de conception de réseaux de cœur 12
1.3.1. Le routage 12
1.3.2. L'affectation de longueurs d'onde 13
1.3.3. Le groupage 14
1.3.4. La protection 15
1.4. Critères d'optimisation 18
1.4.1. Minimiser la probabilité de blocage 18
1.4.2. Minimiser la charge 19
1.4.3. Minimiser la taille des brasseurs 19

Chapitre 2. Algorithmes de routage 21
2.1. Routage et multiflot 21
2.1.1. Le multiflot 22
2.1.2. Application du multiflot au routage 23
2.1.3. Extraire des chemins d'un multiflot entier 24
2.2. Heuristiques de génération et d'extraction de chemins 26
2.2.1. Génération de chemins 26

2.2.2. Extraction des chemins 28
2.3. Méthodes d'approximation du multiflot 29
2.3.1. Arrondi aléatoire du multiflot fractionnaire 30
2.3.2. Approximation du multiflot fractionnaire 31
2.3.3. Amélioration par les plus courts chemins incrémentaux 32
2.4. Routage et affectation de longueurs d'onde 34
2.5. Résultats expérimentaux 36
2.5.1. Génération de chemins 36
2.5.2. Routage des requêtes par un multiflot entier 37
2.5.3. Approximation du multiflot fractionnaire 37
Conclusion 38

Chapitre 3. Algorithmes de groupage 41
3.1. Le groupage 41
3.1.1. Le groupage dans les réseaux WDM 41
3.1.2. Le modèle générique du groupage 42
3.1.3. Modèle de "tubes" 43
3.2. Différentes approches du problème 44
3.2.1. Topologies régulières 44
3.2.2. Topologies quelconques 44
3.2.3. Heuristiques de groupage 45
3.3. Algorithme glouton de *groupage local* 46
3.3.1. Vers un modèle récursif 46
3.3.2. Groupage local d'un nœud 46
3.3.3. Groupage résultant du groupage local 48
3.4. Le routage pour le groupage 49
3.4.1. L'influence du routage sur le groupage 49
3.4.2. Routage *Biggest first* et *Filling threshold* 50
3.4.3. Routage *Filter* 50
3.5. Résultats expérimentaux 52
3.5.1. Performances 52
3.5.2. Résultats de groupage 53
3.5.3. Robustesse à la charge 53
Conclusion 54

Chapitre 4. Algorithmes de protection 57
4.1. Algorithmes connus pour la restauration et la protection 57
4.2. Présentations détaillés d'algorithmes de protection 59
4.2.1. Reroutage global 59
4.2.2. Modélisation de la protection $1:1$ en programmation linéaire 60
4.2.3. Algorithmes de calcul de protection $M:N$ 62
4.3. Un programme linéaire de protection $M:N$ 63
4.3.1. Modèle de Baroni, Bayvel et Gibbens 63
4.3.2. Partage des ressources de protection 65
4.3.3. Limitations du modèle 66
4.4. Modèle mixte 66
4.4.1. Routes principales 67
4.4.2. Flot de protection 67

4.4.3. Exemple d'un jeu de variables pour les programmes linéaires 9 et 10 68
4.4.4. Contraintes de capacité 68
4.4.5. Fonction objectif 68
4.5. Relaxation du modèle et arrondi aléatoire 70
4.5.1. Arrondi aléatoire du modèle de la section 4.4 70
4.5.2. Qualité de l'arrondi aléatoire pour un multiflot 71
4.5.3. Qualité de l'arrondi aléatoire pour l'algorithme 9 73
4.5.4. Programme linéaire final 73
4.6. Résultats expérimentaux 74
4.6.1. Choix expérimental de la taille des ensembles de chemins 74
4.6.2. Comparaison des algorithmes *BBG* et *LSV* 75
4.6.3. Évolution avec la multiplicité 75
4.6.4. Interprétation des résultats 76
Conclusion 77

Chapitre 5. Allocation de fréquence dans les réseaux satellitaires 79
5.1. Modélisation 79
5.1.1. Modèle 79
5.1.2. Problèmes connexes 80
5.1.3. Réutilisation spatiale 80
5.1.4. Niveau d'interférence 81
5.1.5. Types de terminaux et demande 82
5.2. Exemple de problème 83
5.2.1. Etude d'un cas simple 83
5.2.2. Etude d'un cas plus complexe 84
5.3. Résolution du problème d'interférence 86
5.3.1. Calcul du niveau d'interférence 86
5.3.2. Heuristique de génération de familles valides 86
5.4. Résolution des contraintes de placement des capacités sur le canal radio 87
5.4.1. Preuve de la faisabilité du placement 87
5.4.2. Algorithme de placement 88
5.5. Satisfaction de la demande 90
5.5.1. Programme linéaire $\mathcal{P}$ 90
5.5.2. Assignation optimales des types aux familles 91
5.6. Structuration des étapes de l'algorithme 92
5.6.1. Le programme esclave 93
5.6.2. Le programme maître 93
5.6.3. Solution entière de $\mathcal{P}$ 93
5.6.4. Résumé de l'algorithme d'allocation 94
5.7. Résultats expérimentaux 95
5.7.1. Résultats avec 8 spots 95
5.7.2. Résultats avec 32 spots 96
Conclusion 97

Conclusion et perspectives 99

Deuxième partie . Contributions logicielles pour la conception de réseaux 101

Introduction 103

Chapitre 6. Porto 105
6.1. Les données du problème traité dans Porto 105
6.2. Architecture logicielle 106
6.2.1. Découpage fonctionnel 106
6.2.2. Un système de plugin 107
6.3. Algorithmes implémentés dans PORTO 108
6.3.1. Routage 108
6.3.2. Groupage 108
6.3.3. Protection 109
6.3.4. Fonction objectif 109
6.3.5. Autres plugins 109
6.4. Modèle de données détaillé de PORTO 110
6.4.1. Origine des données et formalisation 110
6.4.2. Description des requêtes dans le modèle de données 111
6.4.3. Intégration de l'allocation des ressources 111
6.5. Extraits de résultats expérimentaux 113
Conclusion 115

Chapitre 7. Mascopt 117
7.1. Présentation 117
7.1.1. L'expérience de PORTO 117
7.1.2. Spécificité de Mascopt 118
7.1.3. Une bibliothèque de manipulation de graphes 118
7.2. Modèles objets 119
7.2.1. Le modèle de graphe 119
7.2.2. Le modèle de réseau 122
7.3. Architecture 123
7.3.1. Packages 123
7.3.2. Description des classes principales 124
7.4. Manipulation des objets 125
7.4.1. Système de valuation 125
7.4.2. Information préconstruite en interne 126
7.4.3. Ensembles 126
7.4.4. Généricité et *design pattern factory* 127
7.4.5. Partage des données 127
7.4.6. La cohérence par système de messages 129
7.5. Une courte étude de cas 129
7.6. Sorties fichier 130
7.6.1. Le format MGL 130
7.6.2. La vision DOM du XML 131
7.6.3. Comment étendre le format : l'exemple du MGX 132
7.7. L'interface graphique 133
7.8. Perspectives 134

Conclusion et perspectives 137

Bilan 139

Bibliographie 141

Annexe A. Les réseaux optiques WDM 149
A.1. Exemples de pannes optiques 149
A.2. Réseaux d'étude 150
A.2.1. Réseau R_1 151
A.2.2. Réseau R_2 152
A.2.3. Réseau R_3 153
A.2.4. Réseau NSFNET 154
A.2.5. Réseau EU 155
A.2.6. Réseau $GRID_{35}$ 156
A.2.7. Réseau $GRID_{12}$ 157

Annexe B. Porto 159
B.1. Choix de la plate-forme et des langages 159
B.2. Interface principale 159
B.3. Exemple de déroulement d'un calcul de routage 160
B.3.1. Exécution d'un algorithme d'optimisation 160
B.3.2. Visualisation du résultat 162
B.3.3. Format de données de sortie 163

Annexe C. Mascopt 167
C.1. Multiflot avec Cplex 167
C.2. DTD 170
C.3. Diagramme UML de certains packages de Mascopt 173
C.3.1. Diagramme des classes abstraites de graphes 173
C.3.2. Diagramme des classes non abstraites de graphes 174

Glossaire des sigles

ADM *Add/Drop Multiplexer* : multiplexeur permettant d'ajouter ou de supprimer des flux provenant de la couche électronique dans un nœud, page 8.

ADSL *Asynchronous Digital Subscriber Lines* : réseau de raccordement numérique asymétrique, page 7.

APP Agence pour la Protection des Programmes, page 118.

B-OXC *Band Optical Crossconnect* : brasseur optique de bandes, page 9.

BBG Algorithme de protection adapté à partir de [**BBG99**], page 74.

DOM *Document Object Model* : modèle d'objet de document, page 132.

DTD *Document Type Definition* : définition de type de document, page 130.

F-OXC *Fiber Optical Crossconnect* : brasseur optique de fibres, page 9.

GFC *Graph Foundation Classes for Java*, page 119.

GTL *Graph Template Library*, page 118.

HXC *Hierarchical Crossconnect* : brasseur hiérarchique, page 9.

JDSL *Data Structures Library in Java*, page 119.

LAN *Local Area Network* : réseau local d'entreprise, page 7.

LSV Algorithme de programmation linéaire mixte 13 suivi de l'algorithme 9 d'arrondi aléatoire, page 74.

MFTDMA *Multi-Frequency Time-Division Multiple Access* : accès multiple à répartition dans le temps commuté par satellite à modulation de fréquence, page 79.

MGL *Mascopt Graph Library* : format de fichier de MASCOPT, page 130.

MGX *Mascopt Graph eXtended* : format de fichier étendu de MASCOPT, page 133.

MNH *Minimum Number of Hops* : nombre minimum de sauts, page 26.

NMS *Network Management System* : système de gestion de réseau, page 18.

OADM *Optical Add/Drop Multiplexer* : multiplexeur permettant d'ajouter ou de supprimer des flux optiques dans un nœud, page 8.

OXC *Optical Crossconnect* : brasseur optique, page 9.

RG Algorithme de Reroutage Global, page 74.

RWA *Routing and Wavelength Assignement* : routage et affectation de longueurs d'onde, page 13.

SAX *Simple* API *for* XML : interface de programmation d'applications simple pour le XML, page 132.

SDH *Synchronous Digital Hierarchy* : schéma de multiplexage défini par l'Union Internationale des Télécommunications qui définit les conteneurs STM-1 utilisés dans le transport sur les réseaux optiques (Europe), page 14.

SONET *Synchronous Optical Network* : schéma de multiplexage défini par l'Union Internationale des Télécommunications qui définit les conteneurs STM-1 utilisés dans le transport sur les réseaux optiques (USA), page 14.

SSSP *Single Source Shortest Paths* : plus court chemin à partir d'une source unique, page 31.

STL *Standard Template Library*, page 118.

STM *Synchronous Transport Mode* : conteneur de la SDH correspondant à 155 Mbit/s., page 105.

TDMA *Time-Division Multiple Access* : accès multiple à répartition dans le temps commuté par satellite, page 80.

USSSP *Updated Single Source Shortest Path* : algorithme de mise à jour du plus court chemin à partir d'une source unique, page 34.

W-OXC *Wavelength Optical Crossconnect* : brasseur optique de longueurs d'onde, page 9.

WAN *Wide Area Network* : Réseau étendu, page 7.

WDM *Wavelength Division Multiplexing* : multiplexage en longueur d'onde, page 8.

XML *eXtended Markup Language* : langage de balisage extensible, page 110.

Table des figures

1.1 Choix possibles de commutation optique d'une longueur d'onde 9
1.2 Schéma détaillé d'un brasseur à 3 niveaux. 11
1.3 Un plongement des requêtes. 13
1.4 Exemple d'affectation de longueurs d'onde. 14
1.5 Multiplexage de longueurs d'onde et de bandes dans des fibres. 15
1.6 Protection $1:1$ d'une requête AE pour la panne du câble AB. 17
1.7 Protection $1:1$ d'une requête AE pour la panne du câble CE. 17
1.8 Protection $2:2$ d'une requête AE de taille 2 17

2.1 Flot de s à t et extraction optimale du nombre minimum de chemins. 25
2.2 Représentation d'un réseau avec brassage du flot en v. 25
2.3 Exemple de décomposition pour $z = (s,t)$ et $\mu = 0$. 27
2.4 Marche aléatoire de s à t et $\mu = 1$. 28
2.5 Exemple de graphe en couche, pour 4 requêtes issues de 2 sources et $W = 3$. 35
2.6 Nombre total de longueurs d'onde allouées dans R_1 par *BBG* 37
2.7 Temps de calcul pour différents ϵ sur EU. 38
2.8 Valeur du flot maximum en fonction du nombre de longueurs d'onde. 38
2.9 Temps de calcul pour $\epsilon = 0.05$ sur NSFNET 39

3.1 Comparaison de deux groupages sur un niveau 42
3.2 Séquence d'allocation des longueurs d'onde 47
3.3 Configuration du brasseur pour l'exemple de la figure 3.2. 48
3.4 Exemple de séquence de groupages locaux. 49
3.5 2 stratégies de routage. 49
3.6 UW avec 100% des requêtes. 53
3.7 UW avec 50% des requêtes. 53
3.8 UW pour R_1 en fonction de l'augmentation de la charge. 54
3.9 UW pour R_3 en fonction de l'augmentation de la charge. 54
3.10 UW pour une grille 4x4 en fonction de l'augmentation de la taille des requêtes. 54
3.11 UW pour une grille 4x4 en fonction de l'augmentation du nombre de requêtes. 54

4.1 Exemple de reroutage global pour la requête (s, t) 59
4.2 Partage de la capacité avec le même chemin de longueur d'onde 65
4.3 Partage de la capacité sur un lien k en cas de panne sur l'arc j 66
4.4 Variables du programme linéaire pour la requête $z = (A, E)$ 69
4.5 Nombre total de longueurs d'onde allouées en fonction de W pour R_1 76
4.6 Nombre total de fibres allouées en fonction de W pour R_1 76

5.1 Configuration spatiale des spots 81
5.2 Configuration spatiale pour une unique couleur 81
5.3 Interférences générées par un terminal de la zone z sur un terminal du spot s'. 82
5.4 Petit exemple sur 3 spots. 83
5.5 Ensemble des configurations d'émission possibles pour un seuil à 0.30. 84
5.6 Étapes de l'algorithme de placement 89
5.7 Un exemple d'arrangement global des familles 89
5.8 Modélisation des contraintes de zones par un flot. 91
5.9 Recherche élaguée de familles améliorantes. 94
5.10 Schéma récapitulatif de l'algorithme d'allocation 95
5.11 Un exemple de résultat d'allocation de ressources. 96
5.12 Valeurs d'aire obtenues sur l'ensemble des seuils possibles. 97
5.13 Aires obtenues pour différents nombres de familles aléatoires. 98
5.14 Temps de calcul obtenus pour différents nombres de familles aléatoires. 98

6.1 Découpage fonctionnel de l'outil PORTO 107

7.1 Comparaison avec différentes bibliothèques de graphe 119
7.2 Diagramme de classe de la partie graphe de MASCOPT 121
7.3 Un graphe G_0 valué 122
7.4 Partage des nœuds et des arêtes entre les graphes $G1$ et $G2$ 128
7.5 Partage de l'ensemble de nœuds $VS1$ avec deux ensembles d'arêtes, $ES1$ et $ES2$ 128
7.6 Un graphe orienté valué $G3$ 128
7.7 $G0$ et $G3$ avec des valeurs avec contexte 133
7.8 Le réseau R_1 vu dans le visualiseur 134

B.1 Vue graphique du réseau R_1 160
B.2 Fenêtre principale de PORTO et vue de quelques requêtes 161
B.3 Fenêtre de choix des options de routage et de groupage 161
B.4 Choix de l'option de groupage 162
B.5 Choix de la valeur du facteur de fermeture des conteneurs 162
B.6 Fenêtre d'information sur l'exécution du programme 163
B.7 Représentation d'un câble et de ses différents conteneurs 163
B.8 Représentation des connexions dans un nœud du réseau 164

B.9 Vue graphique des chemins 164
B.10 Fichier XML des chemins 164
B.11 Vue graphique des câbles 165
B.12 Fichier XML des câbles 165

C.1 Diagramme UML des classes abstraites de la partie graphe de MASCOPT 173
C.2 Diagramme UML des classes non abstraites de la partie graphe de MASCOPT 174

Plan de la thèse

Dans cette thèse, nous nous intéressons aux problèmes d'optimisation dans les réseaux de télécommunications. Une première contribution consiste à identifier les problèmes spécifiques aux réseaux optiques ou satellitaires, et à présenter des algorithmes pour optimiser l'utilisation des ressources dans ces réseaux. La seconde contribution de cette thèse est de fournir un formalisme et des outils logiciels pour la conception et l'optimisation de tels réseaux. Les publications résultantes de ces travaux sont citées dans ce plan ainsi que lors de l'exposé détaillé des chapitres correspondants.

Cette thèse s'articule autour de deux grandes parties : la première aborde des modèles pour les réseaux optiques et satellitaires et propose des méthodes algorithmiques nouvelles pour optimiser l'allocation ou l'exploitation des ressources dans ces réseaux. Lorsque les méthodes employées ne permettent pas d'exhiber des garanties théoriques à nos résultats, nous validons nos méthodes par des expérimentations rigoureuses utilisant la plupart du temps des données réelles, collectées grâce à nos collaborations avec un opérateur de télécommunication. La deuxième partie présente les travaux logiciels qui ont pu être développés et qui ont non seulement permis une validation expérimentale des travaux théoriques mais ont aussi contribué à la création d'une bibliothèque générale adaptée à la conception et l'optimisation de réseaux.

Ainsi, la première partie débute par la description des réseaux optiques WDM (*Wavelength Division Multiplexing*). Nous décrivons les modèles et les différents problèmes que nous allons traiter. Cette présentation nous conduit ensuite à décrire les critères d'optimisation qui nous paraîssent importants pour les problèmes d'affectation de ressources. Ces problèmes sont liés les uns aux autres et les chapitres suivants les traitent un à un, tout en essayant d'étudier leurs interdépendances.

Ainsi, nous commençons par étudier le problème du routage [**BCLR03, BCL$^+$03**] dans les réseaux optiques, puis nous nous intéressons au groupage du trafic [**LPS03**] dans ces réseaux. Le chapitre 4 est consacré à la sécurisation ou protection des connexions [**LSV05**]. Enfin, le chapitre 5 présente un algorithme dédié à l'allocation de fréquences dans les réseaux satellitaires [**AAG$^+$04, AAG$^+$05a, AAG$^+$05b**]. Ce travail montre comment des techniques similaires à celles qui sont présentées dans les chapitres précédents dans le cadre des réseaux optiques, interviennent dans un problème d'optimisation de ressources utilisant une technologie totalement différente.

La deuxième partie présente les développements logiciels qui ont été entrepris. Le premier chapitre présente le logiciel PORTO [1] (projet RNRT) dont le développement est à présent clos. PORTO est un logiciel dédié à la résolution de problèmes de routage, groupage et protection dans des réseaux optiques utilisant trois niveaux de brassage : longueurs d'onde, bandes et fibres.

[1] Planification et Optimisation des Réseaux de Transport Optiques, `http://www-sop.inria.fr/mascotte/porto/`

Nous montrons quelles sont les spécificités de ce logiciel et comment il implémente une partie des travaux théoriques présentés en première partie. Dans un second chapitre nous décrivons la bibliothèque MASCOPT [2] [**LSV04**], dont le développement fait suite à l'expérience de PORTO. MASCOPT est une bibliothèque d'optimisation spécialisée pour le domaine des graphes et des réseaux, dont l'ambition est de pouvoir servir de support à la programmation de nombreux types de problèmes réseaux, et non plus seulement dans le cadre des réseaux optiques comme PORTO. Lors de la présentation détaillée de MASCOPT, nous explicitons son architecture et justifions nos choix de conception. La description du modèle objet de MASCOPT nous conduit ensuite à présenter comment utiliser MASCOPT pour implémenter les solutions algorithmiques proposées pour nos problèmes de réseaux WDM.

[2] MASCOPT : `http://www-sop.inria.fr/mascotte/mascopt`

Première partie

Algorithmes pour la conception de réseaux de cœur

Introduction

Cette première partie présente nos contributions algorithmiques pour un problème particulier de conception de réseau (*network design*) et d'allocation de ressources dans les réseaux optiques. Souvent, le problème de conception consiste à prévoir les capacités des liens d'un réseau à construire pour un trafic prédit. Ici nous cherchons à planifier l'allocation des ressources pour écouler un trafic donné sur un réseau existant.

Ces problématiques étudiées sont motivées par la croissance de l'utilisation des réseaux, en terme de bande passante et de nombre de sites connectés. Les équipements déployés par les opérateurs doivent permettre de gérer la complexité des connexions entre les clients tout en minimisant le coût global d'exploitation.

Dans ce contexte, cette thèse dégage trois grands types de problèmes auxquels un opérateur est confronté. Le premier problème est le routage des connexions optiques de ses clients au travers d'un réseau de fibres existant. Le second problème est celui du coût des équipements nécessaires aux extrémités des connexions ou dans les nœuds intermédiaires. Quand cela est possible, on groupe ensemble des connexions utilisant des portions de routes communes de manière à ce qu'elles partagent le coût des équipements d'insertion ou d'extraction des flux de données transportés. Enfin, parallèlement à la croissance du trafic, la demande en qualité de service est de plus en plus forte, notamment pour la sécurisation des connexions : l'opérateur doit veiller à protéger chaque connexion d'une éventuelle panne de lien (il existe d'autres types de pannes mais celui-là est l'un des plus étudiés dans le cadre du problème de conception de réseaux).

Pour chacun de ces trois problèmes détaillés dans le chapitre 1, nous présentons le modèle qui permet de le formaliser puis nous introduisons les algorithmes connus qui permettent d'apporter une première solution. Celle-ci n'est pas toujours satisfaisante, soit en terme d'efficacité, soit en terme de facilité de mise en œuvre sur des instances de réseaux réels. Dans la suite de cette première partie, nous proposons de nouveaux algorithmes mieux adaptés à la résolution d'instances de réseaux réels d'un opérateur. Ainsi, il pourra s'agir dans certains cas de proposer des améliorations de méthodes algorithmiques existantes, comme par exemple au chapitre 2 pour le routage à base de multiflot fractionnaire, et dans d'autres cas il s'agira de montrer l'efficacité expérimentale d'heuristiques lorsque les problèmes sont trop difficiles à résoudre quasi-optimalement, par exemple au chapitre 3 pour le cas du groupage. Notre contribution au problème de protection du chapitre 4 se situe entre ces deux approches : nous proposons une amélioration d'un modèle existant, amélioration quantifiée de manière théorique, et nous présentons des heuristiques pour mettre en œuvre de manière pratique cet algorithme de protection.

Le dernier chapitre de cette première partie n'est pas relié à la thématique des réseaux optiques. En effet, il s'agit d'optimiser les ressources en terme d'émission sur des canaux de fréquences dans un réseau satellitaire. Cependant, ce chapitre est relié aux précédents dans le sens où nous utilisons aussi une technique de programmation linéaire comme dans les trois premiers chapitres pour résoudre à nouveau un problème d'allocation de ressources. Nous introduisons un modèle mixte fractionnaire/entier et nous résolvons le problème d'allocation de fréquences à l'aide d'un programme maître/esclave basé sur la technique de génération de colonnes.

Nous avons volontairement choisi dans cette partie de découper notre propos par types de problème et non par rapport aux techniques algorithmiques utilisées. En effet, notre démarche consiste à trouver les meilleures techniques applicables à un problème donné, et non pas à montrer comment tel ou tel résultat théorique peut s'appliquer dans le cadre de la conception de réseaux.

À la suite de la présentation des algorithmes propres à chaque problème, nous présentons les résultats expérimentaux obtenus. Ces résultats sont basés sur plusieurs réseaux différents, dont ceux décrits en annexe A.2 et obtenus par nos collaborations avec FRANCE TÉLÉCOM. L'implémentation pratique de ces résultats n'est pas précisée dans cette partie puisque ces informations sont rapportées dans la seconde partie de cette thèse, dédiée à deux développements logiciels.

CHAPITRE 1

Conception de réseaux de cœur

Dans cette thèse nous étudions les réseaux de cœur ou réseaux *backbone*. Il s'agit de réseaux qui transportent des flux agrégés en mode commutation de circuits. Les réseaux de cœur WAN (pour *Wide Area Network*) sont maillés à l'échelle d'un pays ou d'un continent et permettent de faire transiter des données à très haut débit. Le nombre de points connectés par de tels réseaux peut atteindre plusieurs centaines de sites (réseaux d'opérateurs ou d'organisation comme par exemple GEANT[1]). Ils ont constamment suivi l'évolution technologique permettant l'accroissement des débits de transmission ainsi que des services offerts. Les grands groupes de télécommunication, souvent opérateurs nationaux, sont les principaux acteurs du déploiement et de la gestion de réseaux de cœur. Nous parlerons de clients d'un opérateur, lorsqu'il s'agit de fournir à un interlocuteur de cet opérateur un certain type d'accès à un réseau de cœur.

Il faut noter que les réseaux de cœur relient les réseaux métropolitains. Ces derniers ont souvent une topologie simple (par exemple, une boucle) et connectent les réseaux locaux au niveau d'une ville. Pour sa part, l'utilisateur final n'a en général qu'une vue limitée à son réseau local (LAN pour *Local Area Network*) depuis une entreprise ou bien depuis un accès par une connexion du type ADSL.

Récemment, les opérateurs ont fait converger les différents types de trafic (voix, données, vidéo ou *triple play*) sur un même support physique. Cette convergence a conduit à l'augmentation du trafic agrégé et nécessite d'augmenter la capacité au niveau de ces réseaux. Aussi, les réseaux de cœur, mais aussi les réseaux métropolitains deviennent de plus en plus maillés et complexes : les travaux présentés dans cette thèse peuvent s'appliquer à ces deux classes de réseaux, même si nous parlons principalement des réseaux de cœur.

Les problèmes d'allocation de ressources dans ces réseaux sont justifiés par le prix très élevé des travaux de génie civil lorsqu'on souhaite modifier les infrastructures du réseau. On préfère alors chercher à mieux utiliser les infrastructures existantes.

Dans cette thèse, nous considérons principalement les technologies optiques, largement déployées aujourd'hui. Ces réseaux relient donc des sites à l'aide de fibres optiques connectées par des brasseurs optiques et/ou électroniques. La technologie actuelle sur une fibre optique permet de faire transiter des données à un débit de l'ordre de plusieurs terabits par seconde. Aucun client d'un opérateur n'ayant une activité nécessitant la totalité d'un réseau de cœur, il s'agit pour l'opérateur de déployer un réseau en adéquation avec l'utilisation future de ses clients, puis, lorsque ce réseau est déployé, d'utiliser les ressources de ce réseau au mieux pour satisfaire ses clients.

[1] cf. `http://www.renater.fr`

1.1. Caractéristiques des réseaux WDM

L'opérateur exploitant un réseau de cœur possède un certain nombre de clients à qui il fournit des connexions d'un point à un autre du réseau (qui peuvent être d'autres opérateurs, de grandes entreprises, ...). Ces connexions ne sont pas directement les connexions IP ou MPLS entre les utilisateurs d'un réseau WAN. Il s'agit plutôt de l'agrégation d'un tel trafic, ceci dans le but de constituer des connexions d'une taille suffisante pour utiliser à bon escient des fibres optiques ayant un débit très élevé.

Nous présentons par la suite la terminologie associée aux réseaux à fibres optiques, ainsi que le principe général de leur fonctionnement. Une description détaillée de cette technologie peut être trouvée dans [**GR00, LD02**].

1.1.1. Le multiplexage en longueurs d'onde

Les réseaux à fibres optiques sont constitués de nœuds reliés par des câbles. Un câble contient plusieurs fibres et un nœud interconnecte plusieurs câbles.

Ces réseaux permettent de faire transiter des données à un très haut débit grâce au multiplexage en longueur d'onde (*Wavelength Division Multiplexing* pour WDM). Plusieurs longueurs d'onde différentes peuvent emprunter la même fibre pour transporter différents flux de données sans interférence. Au niveau d'un nœud, il s'agit de pouvoir distinguer dans un même signal lumineux différentes longueurs d'onde : on parle alors de multiplexage et de démultiplexage de ces longueurs d'onde [**BCJ+97**].

Lorsqu'une fibre optique arrive à un nœud du réseau, les différentes longueurs d'onde multiplexées dans cette fibre peuvent être extraites (*drop*) pour être acheminées vers la couche électronique du réseau (par exemple, la couche IP). De même, dans un nœud, on peut insérer (*add*) des flux électroniques dans des longueurs d'onde. Ces longueurs d'onde sont ensuite multiplexées pour transiter sur une fibre. L'équipement spécifique pour réaliser ces opérations est un ADM pour *Add/Drop Multiplexer*.

Par ailleurs, une longueur d'onde extraite d'une fibre optique ne retourne pas forcément vers la couche électronique : il peut s'agir de récupérer cette longueur d'onde pour la multiplexer dans une autre fibre optique, avec d'autres longueurs d'onde. Dans ce cas, l'équipement au niveau du nœud ne fait pas l'interface avec la couche électronique : il est tout optique et permet d'extraire ou d'insérer une longueur d'onde dans une fibre. Un tel équipement s'appelle un OADM pour *Optical Add/Drop Multiplexer*.

1.1.2. Niveau des conteneurs

Les réseaux de cœur sont des réseaux permettant de véhiculer flux agrégés selon une hiérarchie. Pour la mettre en évidence, nous parlerons de conteneur pour désigner un flux d'une fibre optique qui contient d'autres flux (agrégation ou encapsulation), d'un niveau hiérarchique inférieur. On parle de manière générique de *add* et de *drop* pour les conteneurs qui sont introduits ou retirés du réseau dans un nœud.

Concrètement dans un réseau WDM, le conteneur de plus haut niveau est la fibre optique. Dans ce conteneur, on trouve des bandes qui elles-mêmes contiennent des longueurs d'onde. Cette triple hiérarchie de conteneurs est décrite et utilisée dans [**HPS02, LYK+02, YOM03, CAQ04**] et nous détaillons par la suite les équipements des nœuds qui permettent de manipuler cette hiérarchie.

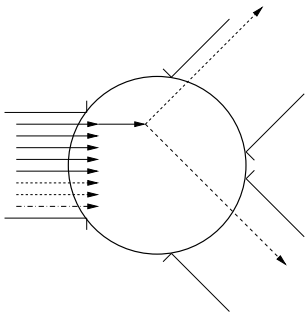


FIG. 1.1 – Choix possibles de commutation optique d'une longueur d'onde

1.1.3. Brassage des conteneurs

Les nœuds d'un réseau WDM assurent une fonction de brassage permettant d'acheminer les conteneurs à travers le réseau jusqu'à destination. Pour bien comprendre ce qu'est le brassage, nous prenons l'exemple du conteneur fibre : lorsque dans un nœud on démultiplexe des longueurs d'une fibre en utilisant un OADM, on peut souhaiter remplacer une ou plusieurs de ces longueurs par d'autres, issues d'une autre fibre, avant de les multiplexer vers une fibre sortant du nœud. On parle alors de brassage de conteneurs puisqu'à partir de plusieurs conteneurs (des longueurs d'onde) entrant dans le nœud on peut échanger ces conteneurs pour une nouvelle répartition de sortie.

Pour désigner de manière générique l'équipement qui permet de réaliser le brassage d'un conteneur on parle d'OXC, pour *Optical Crossconnect* qu'on pourra appeler brasseur optique. Lorsqu'on considère les différents OXC des différents niveaux on regroupe un tel dispositif sous un nom générique : un *brasseur hiérarchique* (hierarchical crossconnect, HXC) [**HSKO99, LYK+02**].

Nous décrivons maintenant plus précisément les différents types d'OXC suivant les niveaux de conteneurs.

1.1.3.1. *Brassage des fibres.* On peut brasser les fibres optiques arrivant de plusieurs câbles dans un nœud : on parle alors de F-OXC pour *Fiber Optical Crossconnect*. Une fibre contenant un certain nombre de longueurs d'onde[2], le brasseur manipule directement toutes ces longueurs d'onde dans un ensemble global non dissocié (le conteneur "fibre").

1.1.3.2. *Brassage des bandes.* Il existe un niveau intermédiaire entre la longueur d'onde simple et la fibre de longueurs d'onde : la bande de longueurs d'onde[3], introduite pour la première fois dans les réseaux en anneau [**GRW00, SS99**]. L'équipement de brassage correspondant au niveau du nœud est logiquement appelé B-OXC pour *Band Optical Crossconnect.*

1.1.3.3. *Brassage des longueurs d'onde.* L'équipement qui permet de réaliser ce brassage de longueurs d'onde est un W-OXC pour *Wavelength Optical Crossconnect*. Cet équipement permet de *commuter* une longueur d'onde entrante dans un nœud vers une longueur d'onde sortante allant vers l'un des nœuds voisins, comme représenté en figure 1.1, où l'on schématise un nœud ayant un câble en entrée et deux câbles possibles en sortie. Cette commutation est réalisée par un commutateur optique.

[2] de 32 à 256 longueurs d'onde par fibre dans les hypothèses de PORTO.

[3] de 8 à 24 longueurs d'onde par bande dans les hypothèses de PORTO.

1.1.4. Conversion de longueurs d'onde dans un réseau tout optique

Dans un réseau tout optique, on requiert la continuité des longueurs d'onde le long d'une suite de fibres. L'absence de conversion de longueurs d'onde signifie que si l'on emprunte une longueur d'onde donnée dans une fibre du réseau, on continuera à utiliser cette même longueur d'onde sur tout le chemin optique emprunté. Ainsi, si on identifie cette longueur d'onde par une couleur dans une fibre, lorsque cette longueur d'onde est brassée par un W-OXC, elle continuera à utiliser la même couleur dans la fibre sortante.

Cependant, les équipements des nœuds du réseau peuvent avoir la capacité de convertir les longueurs d'onde : cela signifie qu'un nœud du réseau peut réémettre une longueur d'onde sur une couleur différente. En général, la conversion est assurée par un passage de l'optique vers l'électronique et vice versa. Certains de nos modèles prennent en compte cette hypothèse.

À ce niveau, il est important de noter que suivant les modèles considérés, le réseau peut être un réseau *tout optique* ou non. Un réseau tout optique permet de faire transiter les longueurs d'onde au travers des nœuds en les brassant avec des OXC. Lorsqu'il s'agit de faire entrer ou sortir un signal électronique, on passe par un OADM qui permet de multiplexer ces signaux dans des flux optiques de différents niveaux.

1.1.5. Représentation des fonctionnalités des brasseurs

L'encapsulation dans différents niveaux hiérarchiques permet de regrouper des conteneurs d'un niveau donné dans des conteneurs de plus haut niveau. Au niveau des nœuds, on doit pouvoir gérer ces différentes encapsulations dans des brasseurs. 8 longueurs d'onde peuvent par exemple être encapsulées dans une bande en passant d'un W-OXC à un B-OXC. Un nœud peut aussi servir de point d'entrée ou de sortie à des données sur le réseau (*add*/*drop*).

La figure 1.2 présente le modèle détaillé du fonctionnement du brasseur. Les termes *fibers, bands, waves* correspondent à fibres, bandes et longueurs d'onde. La fonctionnalité *add* ou *drop* d'un brasseur lui permet d'insérer ou de retirer un signal du réseau. On peut directement insérer une fibre, une bande ou une longueur d'onde dans un des niveaux F-OXC, B-OXC ou W-OXC. Les capacités de multiplexage/démultiplexage sont illustrées par les connexions entre les niveaux deux par deux et par les termes "mux" et "demux". Notons aussi que ce schéma introduit la notion de nombre de ports de multiplexage, c'est-à-dire le nombre de conteneurs d'un niveau pouvant être envoyés au niveau inférieur (et vice versa). Les équipements de brassage fournis par les équipementiers peuvent en effet varier suivant la taille, à fonctionnalités équivalentes. Il y a alors un lien étroit entre la capacité d'un équipement et son coût dont nous reparlons en section 1.4.

La figure 1.2 donne un exemple de brassage de deux fibres entrantes, représentées en noir. On extrait de ces deux fibres les bandes au niveau B-OXC, et si l'on suppose que le nombre de bandes extraites n'est pas supérieur à la capacité d'une fibre, on peut alors les regrouper dans une même fibre avant de remonter au niveau F-OXC. La fibre sortante (en noir) peut alors poursuivre son chemin vers un autre nœud du réseau.

Dans la suite de cette thèse, on utilisera le terme WDM de façon générique. Dans la pratique on distingue les technologies C-WDM, DWDM, UDWDM qui s'appliquent à différents types de réseau et ont des propriétés différentes, notamment en terme de capacité. Le nombre de longueurs d'onde par fibre peut varier de 8 à 400 environ. Dans nos implémentations pratiques, comme dans PORTO, les hypothèses retenues permettent d'encapsuler 8 longueurs d'onde dans une bande et 4 bandes dans une fibre, et par conséquent 32 longueurs d'onde au total par fibre.

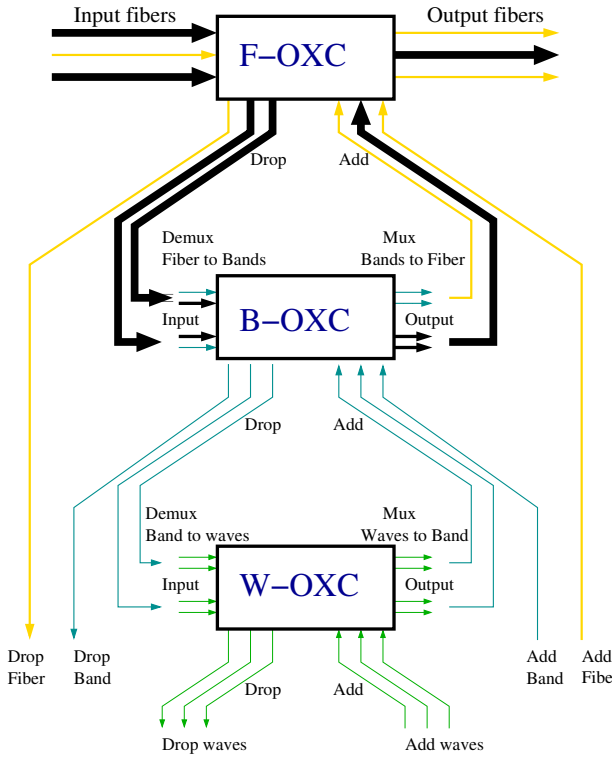


FIG. 1.2 – Schéma détaillé d'un brasseur à 3 niveaux.

1.2. Modèle pour les réseaux WDM

Si l'on considère un réseau WDM déjà opérationnel, ses caractéristiques topologiques sont connues et peuvent être exploitées pour la construction de son modèle. Nous modélisons un réseau WDM par un graphe orienté $G = (V, E)$ où V est l'ensemble des nœuds du réseau et E est l'ensemble des arcs orientés, c'est-à-dire les fibres du réseau. Un arc e du réseau est valué par $c(e)$: c'est la capacité de ce câble optique, exprimé en nombre de fibres.

1.2.1. Matrice de trafic

On parle de matrice de trafic ou d'ensemble de requêtes pour désigner les besoins en trafic entre les couples de sommets (s, t) de $V \times V$ spécifiant les demandes d'acheminement des données des clients du nœud s au nœud t. On notera Z l'ensemble des requêtes à satisfaire. Pour $z = (s, t) \in Z$, on notera $size(z)$ le nombre de longueurs d'onde à faire transiter de s à t dans le réseau. On décomptera donc les requêtes en nombre de longueurs d'onde à faire transiter. La plus petite requête possible (requête unitaire) doit acheminer une longueur d'onde de s à t.

En général, l'ensemble des demandes de la matrice de trafic est représenté par un graphe orienté de requêtes $R = (V, Z)$. On préférera, dans les autres chapitres de cette thèse, utiliser la notation Z comme l'ensemble des arcs du graphe orienté de requêtes, plutôt que R. Chaque requête z de Z est alors valuée par sa taille. Notons enfin que la matrice de trafic est statique et n'évolue donc pas avec le temps. Cette matrice est totalement connue à l'avance. Le routage présenté en section 1.3.1 est donc lui aussi statique.

1.2.2. Multiplicité en longueurs d'onde

La hiérarchie présentée en section 1.1 nous conduit à définir W comme la multiplicité d'un conteneur par rapport à un autre. Nous définissons $W_{x,y}$ comme le nombre de conteneurs de types y que l'on peut encapsuler dans un conteneur de type x. Dans le cas des réseaux WDM nous avons :

Définition 1. *$W_f = W_{f,w}$ est la multiplicité d'une fibre en longueurs d'onde.*

Définition 2. *$W_b = W_{b,w}$ est la multiplicité d'une bande en longueurs d'onde.*

Dans les exemples donnés en section 1.1, nous avions $W_f = 32$ et $W_b = 8$, sachant que $W_{f,b} = 4$.

La multiplicité W est une constante de nos instances : ce paramètre est le même pour tous les câbles du réseau, même si chaque câble n'a pas le même nombre de fibres. Certains travaux proposent des modèles où W varie suivant les câbles du réseaux [**YRL98**].

1.3. Les problèmes de conception de réseaux de cœur

Nous présentons dans cette section trois grands types de problèmes d'allocation de ressources dans les réseaux optiques. Nous précisons quels sont les enjeux de ces problèmes en omettant de préciser les méthodes de résolutions possibles, qui seront présentées plus précisément dans le chapitre correspondant à chaque problème. D'autre part, les différents critères d'optimisation possibles font l'objet d'une section séparée (section 1.4).

1.3.1. Le routage

Le routage des requêtes sur le réseau WDM est le problème le plus simple à identifier dans l'optimisation des ressources réseaux. À une requête, on cherche à associer un ou plusieurs chemins de longueurs d'onde. Un chemin de longueurs d'onde est une suite d'arcs du réseau reliant la source s au puits t de la requête $z = (s, t)$ permettant d'acheminer une certaine quantité de longueurs d'onde.

Ainsi, le routage consiste en un plongement du graphe orienté de requêtes R dans le graphe orienté du réseau G. Ce plongement définit une ou plusieurs routes pour chaque requête $z \in Z$. Lorsqu'il ne s'agit pas d'un problème de conception pur (*network design*), ce routage tient compte des capacités de chaque câble du réseau, sinon on cherche à minimiser la congestion des liens pour pouvoir affecter au mieux les capacités d'un réseau à construire.

La figure 1.3 présente un exemple routage. Le graphe orienté $R = (V, Z)$ représente les requêtes entre certains nœuds du réseau. En dessous, le graphe orienté G représente le réseau physique, avec tous les liens disponibles. Entre s et t, une requête de 7 longueurs d'onde est représentée par un arc en gras. Le routage de cette requête est représenté sur le graphe de câble par une suite d'arcs allant de s à t constituant un chemin (lui aussi en gras).

Enoncé 1. *Etant donné le graphe orienté G du réseau et le graphe orienté R des requêtes, réaliser un plongement de R dans G pour déterminer les routes de chaque requête $z \in Z$ sous des contraintes de capacité.*

Dans cette thèse, nous considérons le routage statique, c'est-à-dire répondant à une certaine quantité de requêtes qui n'évolueront pas dans le temps. Certains aspects dynamiques peuvent apparaître dans le problème de protection décrit en section 1.3.4 puisqu'il s'agit d'envisager des cas de pannes possibles sur le réseau. Le routage traité ici est dit *offline* car l'ensemble

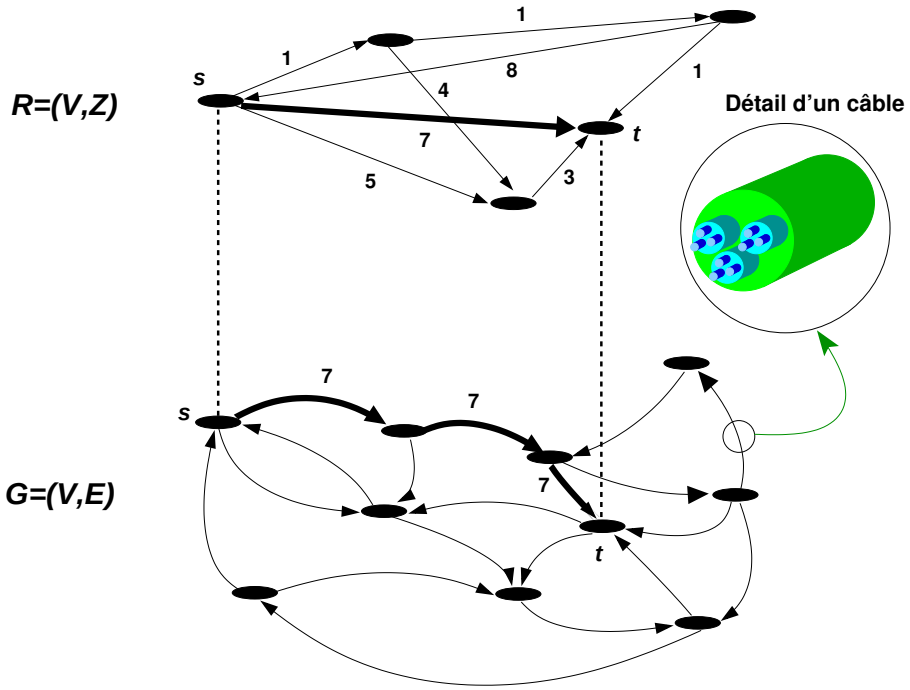


FIG. 1.3 – Un plongement des requêtes.

des requêtes est connu à l'avance ; a contrario, un routage *online* répond dynamiquement à un ensemble de requêtes arrivant en temps réel sur le réseau [**YLR99**].

1.3.2. L'affectation de longueurs d'onde

L'affectation de longueurs d'onde, ou RWA pour *Routing and Wavelength Assignement* lorsque le routage est réalisé en même temps, est un autre point de vue combinatoire du routage optique mais aussi du groupage que nous présentons dans la section qui suit. La résolution de ce problème permet de choisir sur quelle longueur d'onde un chemin circule dans les câbles. S'il n'y a pas de conversion possible dans les nœuds, un chemin utilisera la même longueur d'onde, numérotée 1 par exemple, pour traverser tout le réseau. Autrement dit, une route empruntant la longueur d'onde numéro 1 d'une fibre ne pourra pas, dans une autre fibre, passer sur la longueur d'onde numéro 2. Dans ce cas, il s'agit d'affecter la longueur numéro 1 à une route donnée, c'est-à-dire à une unité parmi $size(z)$ d'une requête $z \in Z$. L'affectation de longueurs d'onde dépend très fortement de la phase de routage comme le précise [**BPS02, HV98, BHP98**]. Par ailleurs, même si l'on fixe le routage, le problème d'affectation de longueurs d'onde reste $\mathcal{NP}$-complet, car il se ramène à un problème de coloration [**BS97**].

Lorsque l'on introduit la possibilité de convertir des longueurs d'onde à certains nœuds du réseau, il faut alors décider en plus de l'affectation des conversions effectuées qui restent statiques avec le routage. Pour des conversions dynamiques (routage *online*), il faut décider des conversions possibles en ne connaissant pas les requêtes à l'avance. Lors de l'arrivée des requêtes, il faut une politique d'assignation/conversion qui va permettre de créer le chemin et ainsi que les choix d'affectation et de conversion [**YLR99**].

L'exemple de la figure 1.4 montre trois nœuds A, B et C par lesquels transitent 3 requêtes (1 unité de trafic chacune). On affecte 2 unités de trafic sur les longueurs d'onde numéro 1

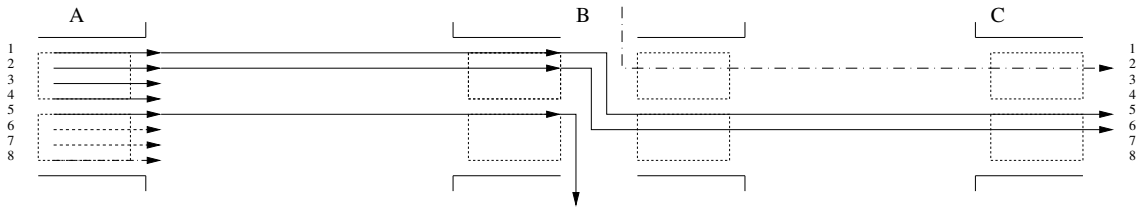


FIG. 1.4 – Exemple d'affectation de longueurs d'onde.

et 2 du nœud A, et ce, jusqu'au nœud B. On convertit alors ces longueurs d'onde pour une affectation au nœud C sur les longueurs d'onde numéro 5 et 6. Notons enfin que la longueur d'onde numéro 5 au nœud A suit un routage différent (elle ne se rend pas en C sur l'exemple).

1.3.3. Le groupage

Nous appelons problèmes de groupage, les problèmes connus sous le nom de *traffic grooming*. Le problème du routage, tel qu'il est exprimé précédemment, ne prend pas en compte les équipements des nœuds du réseau. Intuitivement, le terme de "groupage" signifie que l'on regroupe des conteneurs d'un niveau inférieur dans un conteneur de niveau supérieur le long d'un chemin acheminant une requête dans le réseau.

Etant donné la hiérarchie des conteneurs décrite en section 1.1.2 et montrée dans le détail d'un câble de la figure 1.3, il n'est pas équivalent de faire transiter 1 bande ou 8 longueurs d'onde de s à t. En effet, dans les nœuds intermédiaires, il faudra brasser dans le premier cas 1 bande dans un B-OXC et, dans l'autre cas, 8 longueurs d'onde dans un W-OXC. Si l'on suppose que les équipements au niveau des nœuds sont à prévoir (ou à configurer), le choix du regroupement de conteneurs de plus petits niveaux dans des conteneurs de plus haut niveau influe sur l'équipement des nœuds et donc sur le coût d'exploitation ou de mise en place du réseau.

Si l'on considère que le routage des requêtes est connu, le groupage (*grooming*) consiste donc à déterminer la taille des équipements, W-OXC, B-OXC, F-OXC, au niveau des nœuds du réseau qui vont supporter les routes de longueurs d'onde qui transitent de s à t ainsi que les groupements de conteneurs qui traversent ce réseau. Dans chaque nœud il faudra décider quelles longueurs d'onde, bandes, fibres sont entrantes ou sortantes du réseau. Il faudra aussi décider s'il doit y avoir groupage ou dégroupage de conteneurs de différents niveaux, comme présenté en figure 1.1.

Le problème du groupage a été largement étudié dans d'autres hypothèses, notamment pour le groupage de flux SDH/SONET. Dans [**CM00, WCVM01**] par exemple, les auteurs considèrent un ensemble de requêtes complet avec $Z = (V \times V)$, c'est-à-dire un trafic *All-to-All* que l'on doit grouper sur un réseau en anneau.

Nous présentons en figure 1.5 un exemple de groupage. Nous ne considérons qu'une partie du réseau composé de 3 nœuds A, B, C connectés par une fibre de A à B et de B à C. On suppose que 3 chemins de longueurs d'onde sont pré-établis :

- une requête d_1 arrivant au nœud A d'un nœud précédent (représentée en pointillés courts) et ayant pour destinataire le nœud C
- une requête d_2 entrant au nœud A dans la couche W-OXC et sortant au nœud C (représentée en pointillés longs)

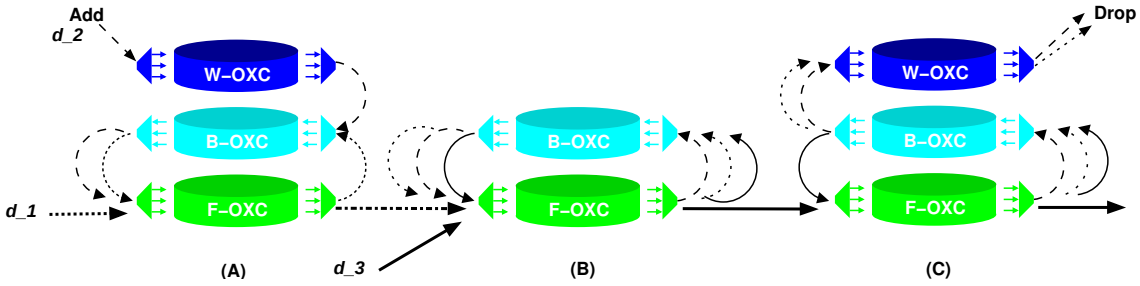


FIG. 1.5 – Multiplexage de longueurs d'onde et de bandes dans des fibres.

– une requête d_3 arrivant au nœud B via une fibre différente de la fibre (AB), et sortant du nœud C pour atteindre un autre nœud du réseau (représentée en trait plein)

Au nœud A, les longueurs d'onde de la demande d_2 sont multiplexées dans une bande dans la couche B-OXC (pointillés). La demande d_1 entre sur la couche F-OXC et remonte au niveau bande (extraction d'une bande de la fibre). d_2 est multiplexée avec d_1 dans la couche B-OXC (flèche pointillée) : Les deux bandes sont ainsi multiplexées sur une même fibre qui retourne au niveau F-OXC.

Au nœud B, une nouvelle demande d_3 arrive d'une autre fibre. On démultiplexe le signal de la fibre issu de A et le signal de d_3 en remontant au niveau B-OXC. On extrait alors les deux bandes issues de A et la bande de d_3 et on multiplexe les 3 bandes dans une seule fibre qui retourne au niveau F-OXC.

Enfin, au nœud C on démultiplexe la fibre arrivant de B en passant au niveau B-OXC. Les bandes de la demande d_1 et d_2 sont encore démultiplexées en remontant au niveau W-OXC : on récupère les longueurs d'onde qui doivent s'arrêter en C (drop). La bande de la demande d_3 est remultiplexée dans une fibre et poursuit son chemin dans le réseau.

Cet exemple présente un agencement des groupes de longueurs d'onde et groupes de bandes qui permettent de faire circuler les 3 demandes sur le réseau. Ces regroupements ont un impact direct sur les multiplexages nécessaires au différents niveaux, W-OXC, B-OXC, F-OXC. Remarquons que dans cet exemple, le nœud B ne nécessite pas d'équipement du type W-OXC. Le problème du groupage est intimement lié au problème du routage présenté en section 1.3.1. On peut se poser le problème du groupage, une fois le routage réalisé, ou chercher à résoudre les deux problèmes conjointement.

Enoncé 2. *Etant donné le graphe orienté G du réseau et le graphe orienté R des requêtes et un plongement de R dans G (routage), déterminer les équipements au niveau des nœuds du réseau et les regroupements de conteneurs dans chaque câble.*

1.3.4. La protection

Dans les réseaux WDM il arrive qu'une interruption de service accidentelle ait lieu, coupant une ou plusieurs routes du réseau. La cause de ces pannes peut être de différentes natures : une coupure physique du câble optique, un incendie dans un local, ou simplement une erreur humaine [**MIB$^+$04**]. Une interruption de service n'est pas assez rare pour qu'elle puisse être considérée comme insignifiante (des exemples de pannes pour un opérateur sont présentés en annexe A.1). Par ailleurs, la quantité de données transportées dans une fibre est telle qu'une

coupure provoque la perte d'une importante quantité de données. On cherche alors naturellement des mécanismes de protection qui rendent le réseau *tolérant aux pannes* ou assurant une *continuité face aux pannes*.

Les pannes considérées seront celles au niveau des câbles du réseau optique, c'est-à-dire des arcs de E. On considérera aussi qu'une seule panne à la fois se produit sur l'ensemble du réseau. Les pannes pouvant survenir au niveau des nœuds ne sont pas prises en compte ici, ceci pour deux raisons : si l'un des ports du nœuds ne fonctionne plus, cela revient à considérer une panne de câble et d'autre part, l'équipement au niveau des nœuds est souvent doublé, pour éviter une panne trop grave [**ACB97**]. On ne considérera pas non plus le cas d'une panne sur une fibre d'un lien, les autres fibres continuant à fonctionner et on préférera prendre le cas plus général d'une coupure totale d'un unique câble du réseau. Cependant, certains travaux récents s'intéressent à l'éventualité d'une double panne dans un réseau optique [**CSC02, SP03**].

Il existe différents types de protections qui peuvent être mis en œuvre, nécessitant des routages différents. Une présentation détaillée et classifiée des différents types de protection sont présentés dans [**FV00**], ainsi que la protection de la couche IP/MPLS. Nous présentons ci-après les différences entre la restauration, la protection dédiée et partagée.

1.3.4.1. *Restauration.* La restauration consiste à rerouter dynamiquement des connexions lorsqu'un cas de panne survient sur le réseau. On doit alors calculer, au moment de la panne, un nouveau routage à partir des ressources disponibles. On parle d'algorithmes *online* (comme pour le routage) puisqu'ils ne répondent pas à un problème statique ou connu à l'avance. Dans la suite de cette thèse nous ne traitons pas le problème de la restauration, mais celui de la protection.

1.3.4.2. *Protection par reroutage global.* Le reroutage global consiste à prévoir un routage admissible pour chaque cas de panne possible. Pour chaque routage, une certaine capacité est nécessaire sur un câble du réseau. On choisit la capacité maximum, pour tous les cas de pannes possibles et l'on choisit d'allouer cette capacité maximale : on obtient l'assurance de pouvoir router, quelle que soit la panne, l'ensemble des requêtes sur le réseau.

L'inconvénient direct d'une telle politique de protection vient du fait qu'entre l'état sans panne et un état de panne donné, aucune garantie n'est donnée quant à l'emplacement des routes principales et des changements à opérer. Dans le pire des cas, toutes les routes principales sont à modifier, provoquant un impact d'ordre technique dans la configuration des nœuds. Même si la protection par reroutage global est une solution envisageable, nous considérons cette solution au problème de protection comme une borne inférieure théorique aux autres méthodologies de protection présentées ci-après.

1.3.4.3. *Protection dédiée et partagée.* La protection dédiée et partagée ne nécessite pas un reroutage total en cas de panne. Il s'agit au contraire de ne rerouter que les chemins principaux touchés par la panne par des chemins de secours. La protection dédiée nécessite d'allouer un chemin de secours qui ne peut être réutilisé dans un autre contexte. À l'inverse, la protection partagée permet d'utiliser une même ressource pour deux chemins de secours qui ne pourraient être activés en même temps. Notons enfin que le reroutage global fait partie de la protection partagée (tout le réseau est partagé).

On distingue alors la classification suivante pour la protection dédiée : la protection $1+1$ qui consiste à envoyer la même information sur deux chemins disjoints (le chemin principal et de protection) en même temps. Au niveau du nœud de destination, le signal est reçu en double, garantissant la réception d'au moins un signal en cas de panne. La protection $1:1$ dédiée

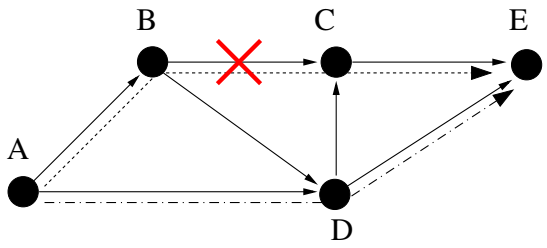


FIG. 1.6 – Protection $1 : 1$ d'une requête AE pour la panne du câble AB.

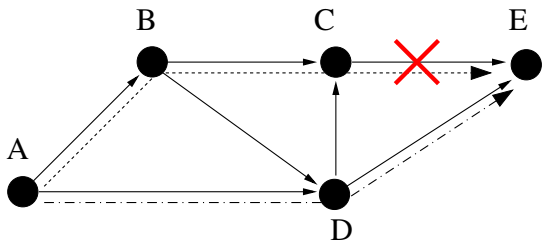


FIG. 1.7 – Protection $1 : 1$ d'une requête AE pour la panne du câble CE.

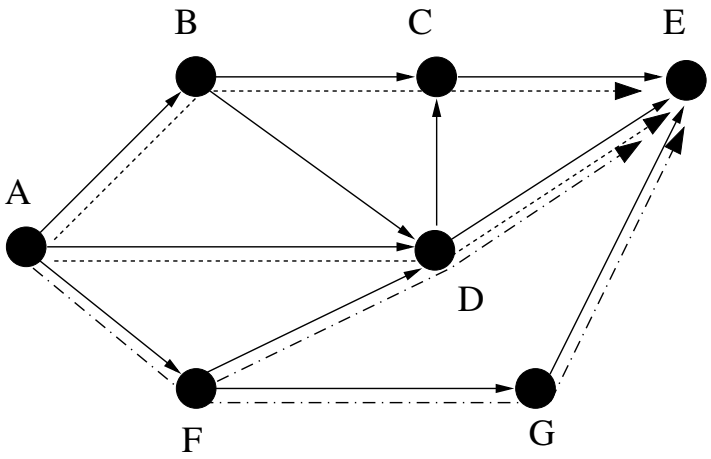


FIG. 1.8 – Protection $2 : 2$ d'une requête AE de taille 2

réserve un chemin de secours pour chaque chemin principal. En cas de panne, le chemin de secours est activé.

Dans le cas de la protection partagée, on parle aussi de protection $1 : 1$. Dans ce cas, les chemins de secours peuvent partager des longueurs d'onde entre eux. Ce cas est montré en figures 1.6 et 1.7 où l'on protège le chemin (A, B, C, E) pour deux cas de pannes possibles. Pour ces pannes, on utilise le même chemin de secours, (A, D, E), qui est dit "partagé".

Plus généralement, pour plus de flexibilité, on utilise la protection $M : N$. Pour une même requête $z \in Z$, M chemins principaux sont protégés par N chemins de secours. Les N chemins de secours peuvent partager des longueurs d'onde avec d'autres chemins de secours (de la même requête ou d'une requête différente) qui ne peuvent s'activer pour la même panne. La figure 1.8 montre le cas d'une protection du type $2 : 2$. Les chemins principaux, en pointillés, sont protégés par les chemins en pointillés discontinus. On note alors que sur le câble AF, on peut partager la capacité de protection puisque les chemins principaux ne peuvent tomber en panne en même temps.

Notons enfin que pour la protection $1 : 1$ et $M : N$, les ressources réservées pour les chemins de protection ne sont pas utilisées. En pratique, les opérateurs font circuler sur ces canaux des flux non prioritaires. Ces flux peuvent être interrompus et remplacés par des flux de protection, le temps que la situation revienne à la normale.

1.3.4.4. *Reconfiguration des brasseurs.* Les différents types de protection possibles n'ont pas le même impact sur la configuration des brasseurs optiques. L'avantage de la protection dédiée réside dans le fait que les brasseurs ne changent pas de configuration en cas de panne. Seul un signal d'activation du chemin de secours doit être envoyé au brasseur du nœud source pour la protection $1:1$.

Pour la protection partagée, la mise en place des chemins de secours peut nécessiter de nombreuses reconfigurations. Les ressources étant partagées entre les chemins de secours, les brasseurs ne peuvent être configurés dans un état donné puisque la participation d'une longueur d'onde à une route peut changer suivant les pannes.

La coordination de la configuration des brasseurs doit donc être centralisée et commandée par un réseau de contrôle (ou éventuellement distribuée). En cas de panne d'un câble, le brasseur connecté à ce câble cesse de recevoir le faisceau : il transmet alors un signal de panne au NMS (*Network Management System*) connecté à tous les nœuds du réseau [**BB97**]. Le NMS envoie alors un ordre de reconfiguration aux brasseurs concernés pour ce cas de panne précis.

De nombreux autres problèmes peuvent se poser dans les réseaux tout optiques, mais nous nous intéresserons qu'aux trois types de problèmes que nous venons de présenter. À partir de ceux-ci, nous précisons par la suite quels critères d'optimisation nous considérons.

1.4. Critères d'optimisation

La résolution des problèmes précédents dépend du critère d'optimisation choisi. Dans cette thèse, nous ne faisons pas les hypothèses de conception de réseau (network design) [**MBR96, SB99**] et nous prenons donc en compte les contraintes de capacité (les câbles du réseau existent déjà). De notre point de vue, nous cherchons en général à optimiser l'utilisation des ressources disponibles, soit pour maximiser l'utilisation de ces ressources, soit pour libérer un maximum de ressources qui pourront être réemployées. Ces notions de ré-utilisation sont liées à des critères de coûts d'exploitation du réseau. On parle de minimiser le coût d'utilisation lorsque l'on peut réduire le taux d'utilisation des câbles du réseau (on libère des longueurs d'onde pour d'autres communications) ou lorsque l'on peut réduire la taille des équipements aux différents nœuds du réseau (des équipements du types W-OXC, B-OXC, F-OXC plus simples sont moins coûteux).

1.4.1. Minimiser la probabilité de blocage

Répondre à un ensemble de requêtes avec un réseau déjà dimensionné peut parfois être impossible si le réseau n'a pas la capacité nécessaire pour supporter l'ensemble de la demande. On parle dans ce cas d'un blocage si une partie des requêtes ne peut être acheminée à son point de destination. Suivant les hypothèses prises en compte, et notamment si l'on considère un routage sur un réseau dynamique, il est intéressant d'étudier la probabilité calculée ou constatée d'une politique de routage et/ou groupage qui pourrait bloquer le réseau à un instant donné [**BK95**].

Les auteurs de [**YLR99**] présentent une étude exhaustive des différents critères qui influent sur cette probabilité, et notamment l'influence de la possibilité de conversion en longueurs d'onde dans les nœuds du réseau. Les critères principaux intervenant, en plus de la topologie du réseau, sont la capacité $c(e)$ et W_f, ainsi que l'arrivée des requêtes si la matrice de trafic est dynamique.

Lorsque le blocage concerne le problème de la protection, il peut s'agir de garantir la plus grande probabilité de non blocage pour l'ensemble des cas de pannes possibles. La minimisation de la probabilité de blocage permet de réduire la capacité allouée pour les chemins de protection puisque l'on relâche le critère de tolérance aux pannes.

La probabilité de blocage ne nous intéressera pas dans cette thèse puisque nous traitons une matrice de trafic statique et que nous cherchons à garantir une solution toujours réalisable. Concernant la protection, nous nous plaçons dans les hypothèses de [**BBG99**], cherchant à garantir 100% des cas de pannes possibles.

1.4.2. Minimiser la charge

Pour le problème de routage, le coût que l'on considère dominant est celui de l'utilisation des câbles du réseau. À chaque longueur d'onde on associe un canal optique qui transporte un flux (de l'ordre de 2.5 Gbps si c'est un STM-16 par exemple). Le nombre de ressources du type longueurs d'onde utilisé par un chemin de routage est donc égal à la taille de ce chemin (en terme de flux) multiplié par sa longueur (en nombre de liens). En effet, on affecte les conteneurs le long des chemins utilisés : la somme totale de ces conteneurs est la charge totale du réseau (*load*).

Dans le cas du modèle hiérarchique à trois niveaux, on peut définir la charge pour un type de conteneur :

Définition 3. *La charge en nombre de longueurs d'onde (respectivement de bandes, fibres) du réseau est notée nb_w (respectivement nb_b, nb_f). C'est le nombre total de longueurs d'onde (respectivement de bandes, fibres) qui sont utilisées (donc non vides).*

Une fibre est comptabilisée dans la charge à partir du moment où une de ses longueurs d'onde est utilisée. Ceci explique pourquoi on a : $nb_f \times W_f \geq nb_w$. On peut aussi définir la charge d'un lien du réseau comme la somme des conteneurs utilisant ce lien. Pour les trois niveaux longueur d'onde, bande et fibre on note la charge d'un lien e : $nb_w(e)$, $nb_b(e)$ et $nb_f(e)$.

1.4.3. Minimiser la taille des brasseurs

Le problème du routage ne cherche pas à optimiser l'équipement des nœuds du réseau. En général, c'est plutôt le groupage qui utilise ce critère puisque l'encapsulation de conteneurs permet d'influencer le nombre de ports du brasseur. Dans [**CM00, WCVM01**] par exemple, on cherche à minimiser le nombre de longueurs d'onde entrantes (*add*) ou sortantes (*drop*) du réseau c'est à dire le nombre de ports des ADM. Dans nos hypothèses, ce nombre est fixe et on se concentre surtout sur la taille des OADM.

Au niveau de l'équipement des nœuds, on aura donc tendance à vouloir quantifier le nombre de conteneurs à brasser dans un niveau donné, en partant du principe que le brassage à réaliser coûte d'autant plus cher que le nombre de conteneurs est important. Il peut aussi s'agir de minimiser le nombre de conversions de longueurs d'onde au niveau des brasseurs, si l'on considère que cette possibilité est limitée ou fait partie du coût d'exploitation du réseau.

Suite à cette présentation des réseaux optiques, nous présentons les travaux de cette thèse relatifs au problème du routage.

CHAPITRE 2

Algorithmes de routage

Pour faire face aux demandes croissantes de trafic, on doit exploiter au mieux les ressources existantes. En effet, les coûts de génie civil pour poser de nouvelles fibres sont très élevés et on cherche donc à utiliser au mieux les fibres présentes sur le réseau. La question de l'optimisation des routes empruntées par les requêtes devient alors un enjeu crucial. En effet, l'affectation des ressources pour le routage influe fortement sur l'ensemble du coût du réseau mais aussi sur d'autres coûts liés au groupage ou à la protection, étudiés dans les chapitres suivants.

Dans ce chapitre on présente plusieurs algorithmes de routage ainsi que des résultats d'expériences démontrant leur qualités. En effet, deux routages différents fournissent le même service mais donnent deux charges différentes et donc des coûts différents. Dans un premier temps, on donne le modèle connu des multiflots entiers du type *sommet-arc* et *arc-chemin*. Ces méthodes sont les bases qui permettent de résoudre des modèles plus complexes prenant en compte la spécificité des réseaux optiques, comme la conversion ou l'affectation de longueurs d'onde. Dans les hypothèses de ce chapitre, nous ne tenons pas compte de la conversion ; en ce qui concerne l'affectation, nous la traitons dans un modèle particulier utilisant des couches pour représenter la couleur d'une longueur d'onde.

Nous introduisons ensuite de nouvelles heuristiques pour l'obtention de routages quasi-optimaux en utilisant le calcul du multiflot fractionnaire. Nous proposons un nouvel algorithme d'approximation du multiflot fractionnaire et nous montrons comment adapter cette approximation au cas d'un réseau optique. Enfin, nous présentons les résultats expérimentaux de ces différentes méthodes.

2.1. Routage et multiflot

Le problème du routage, présenté en section 1.3.1 est un problème largement étudié dans la littérature [**OMV97, LD02**]. Le problème du routage est $\mathcal{NP}$-complet. Lorsque l'on prend en compte l'affectation des longueurs d'onde, le routage reste $\mathcal{NP}$-complet. Les outils pratiques de résolution se tournent alors vers des heuristiques générales avec des méthodes de *recuit simulé*, de *recherche tabou* ou d'*algorithmes génétiques* [**HV98**].

Lorsque la matrice de trafic est régulière[1], les algorithmes de routage optique utilisent des techniques issues de la théorie des graphes, en utilisant par exemple le calcul d'arbres couvrants pour le cas d'un trafic multicast [**ZGM03**]. Si la topologie du réseau est elle aussi régulière, elle permet d'obtenir de bons résultats théoriques, comme par exemple lorsque l'on traite des réseaux en anneaux. Le routage étant trivial, on se ramène à un problème ou l'on cherche à

[1] Par exemple en *All-to-All* où chaque nœud communique avec tous les nœuds du réseau.

minimiser le nombre de couleurs utilisées (RWA), largement étudié dans la littérature [**Tuc75, Kar80, Kum98**].

Dans ce chapitre, nous commençons par nous intéresser à la problématique du routage comme un multiflot non mono-routé sur un graphe. À partir des techniques de calcul d'un multiflot, nous présentons en section 2.4 une modélisation des réseaux optiques qui permet de répondre au problème de l'affectation de longueur d'onde. Ce dernier problème interviendra à nouveau dans le chapitre 3 consacré au groupage. Cette section présente les modèles classiques de programmation linéaire qui permettent de résoudre le problème du multiflot. Ces modèles serviront de base à la résolution de problèmes plus complexes, notamment celui du groupage et de la protection aux chapitres 3 et 4.

2.1.1. Le multiflot

Un flot représente l'écoulement d'une certaine quantité de données d'une source s à un puits t respectant les lois de Kirchoff aux nœuds du graphe (cf. équation 2.2). Chaque flot correspondant à une requête donnée. Un multiflot est un ensemble de flots. Un problème de multiflot acheminant les requêtes $z \in Z$ sur un graphe $G = (V, E)$ se calcule en temps polynomial lorsque le flot est fractionnaire (relaxation linéaire du multiflot entier). Le modèle classique utilisé est le modèle *sommet-arc* qui nécessite une variable de flot par arc et par requête.

Soit $x_{e,z}$ la variable de flot pour l'arc e et la requête z. On note $add(v, z)$ et $drop(v, z)$ le flot entrant et sortant au nœud v pour la requête z. $\delta^+(v)$ et $\delta^-(v)$ sont les ensembles d'arcs sortant et entrant pour le nœud v.

Programme linéaire 1 (multiflot sommet-arc)**.**

$$\forall e \in E, \sum_{z \in Z} x_{e,z} \leq c(e) \tag{2.1}$$

$$\forall v \in V, \forall z \in Z, \sum_{e^- \in \delta^-(v),\ e^+ \in \delta^+(v)} x_{e^-,z} - x_{e^+,z} = drop(v, z) - add(v, z) \tag{2.2}$$

$$\forall e \in E, \forall z \in Z,\ x_{e,z} \geq 0 \tag{2.3}$$

$$\textit{Min} \sum_{z \in Z,\ e \in E} x_{e,z} \tag{2.4}$$

L'avantage d'une telle formulation réside dans le fait qu'elle s'exprime avec un nombre polynomial de variables ($\mathcal{O}(|E||Z|)$) et de contraintes ($\mathcal{O}(|V||Z| + |E||Z| + |E|)$) : les algorithmes de programmation linéaire la résolve en temps polynomial (par exemple, la méthode de l'*ellipsoïde* [**NW88**], chapitre I.6, page 147). Cependant, pour la résolution du problème du routage, le modèle *sommet-arc* se prête mal à la détermination des routes, ces routes étant nécessaires pour le problème du groupage. Nous montrons en section 2.1.3, comment utiliser des heuristiques d'extraction de chemins.

Un deuxième modèle, présenté ici, est le modèle *arc-chemin*. Il consiste à calculer les flots sur les arcs du graphe à l'aide de l'ensemble des chemins possibles pour chaque requête considérée. L'inconvénient de la formulation *arc-chemin* est d'avoir à exprimer tous les chemins possibles pour chaque requête $z = (s, t) \in Z$. La taille de ces ensembles de chemins croît exponentiellement avec la taille du graphe. Pour pallier ce problème, nous présentons en section 2.2 des heuristiques pour générer de bons ensembles de chemins candidats.

On note $\mathcal{P}_z$ l'ensemble des chemins possibles pour la requête z. $\delta^A_{p,z}$ est la variable indiquant la quantité de flot empruntant le chemin $p \in \mathcal{P}_z$ pour la requête z.

Programme linéaire 2 (multiflot arc-chemin)**.**

$$\forall z \in Z,\ \sum_{p \in \mathcal{P}_z} \delta^A_{p,z} = size(z) \tag{2.5}$$

$$\forall e \in E,\ \sum_{z \in Z,\ p \in \mathcal{P}_z,\ e \in p} \delta^A_{p,z} \leq c(e) \tag{2.6}$$

$$\forall z \in Z,\ \forall p \in \mathcal{P}_z,\ \delta^A_{p,z} \geq 0 \tag{2.7}$$

$$\textit{Min} \sum_{z \in Z,\ p \in \mathcal{P}_z,\ e \in p} \delta^A_{p,z} \tag{2.8}$$

Ce programme linéaire possède $\mathcal{O}(|Z||\mathcal{P}_Z|)$ variables et $\mathcal{O}(|Z||\mathcal{P}_Z|+|Z|+|E|)$ contraintes, où $|\mathcal{P}_Z|$ désigne la taille asymptotique des ensembles $\mathcal{P}_z$. Les fonctions objectifs présentées dans ces deux programmes linéaires comptabilisent la taille totale du multiflot allouée sur le graphe. Nous rappelons qu'il s'agit de la somme pour toutes les requêtes et pour tous les chemins de routage utilisés du nombre de longueurs d'onde pour chaque arc du chemin (cf. section 1.4.2).

2.1.2. Application du multiflot au routage

L'utilisation des programmes linéaires du multiflot pour le calcul du routage dans un réseau WDM nécessite plusieurs adaptations. Tout d'abord, les fonctions objectifs doivent prendre en compte les critères d'optimisation présentés en section 1.4. La minimisation du flot total sur le réseau est en soi un objectif légitime mais, la plupart du temps, on souhaite insérer la granularité de l'allocation des fibres du réseau, comme présenté en section 1.4.2. On comptera donc principalement le nombre de fibres allouées dans le réseau et non le nombre de longueurs d'onde.

Si l'on relâche la contrainte d'intégrité des variables, on obtient un programme linéaire fractionnaire. La résolution des programmes linéaires précédents est alors polynomiale. Lorsque l'on souhaite les résoudre en entier, leur résolution devient $\mathcal{NP}$-difficile. De plus, la contrainte d'intégrité amène une autre conséquence, concernant la fonction objectif. Dans un programme linéaire fractionnaire, le nombre de fibres et de longueurs d'onde sont proportionnels, rendant les deux objectifs équivalents. Le routage que l'on cherche à obtenir se modélise par un flot entier, c'est-à-dire que les chemins ou les flots alloués doivent être des résultats entiers allouant des pleines longueurs d'onde sur le réseau. Dans ce cas, la résolution des programmes linéaires précédents est $\mathcal{NP}$-complet et la fonction objectif comptant le nombre de fibres allouées complexifie la recherche de la solution (la fonction objectif varie par "sauts" au cours du processus d'optimisation ; le pas est de W_f à la place de "sauts" d'une seule longueur d'onde).

Introduisons la variable $nb_f(e)$ représentant le nombre de fibres par câble e. Les programmes linéaires présentés en section 2.1.1 deviennent alors :

Programme linéaire 3 (routage sommet-arc)**.**

$$\forall e \in E,\ \sum_{z \in Z} x_{e,z} \ \leq\ W_f \times nb_f(e) \tag{2.9}$$

$$\forall e \in E,\ W_f \times nb_f(e) \ \leq\ c(e) \tag{2.10}$$

$$\forall v \in V, \forall z \in Z,\ \sum_{e^- \in \delta^-(v),\ e^+ \in \delta^+(v)} x_{e^-,z} - x_{e^+,z} \ =\ drop(v,z) - add(v,z) \tag{2.11}$$

$$\forall e \in E, \forall z \in Z,\ x_{e,z},\ nb_f(e) \qquad \textit{entiers} \tag{2.12}$$

$$\textit{Min} \sum_{e \in E} nb_f(e) \tag{2.13}$$

Programme linéaire 4 (routage arc-chemin)**.**

$$\forall z \in Z,\ \sum_{p \in \mathcal{P}_z} \delta^A_{p,z} = size(z) \tag{2.14}$$

$$\forall e \in E,\ \sum_{z \in Z,\ p \in \mathcal{P}_z,\ e \in p} \delta^A_{p,z} \ \leq\ W_f \times nb_f(e) \tag{2.15}$$

$$\forall e \in E,\ W_f \times nb_f(e) \ \leq\ c(e) \tag{2.16}$$

$$\forall z \in Z,\ \forall p \in \mathcal{P}_z,\ \delta^A_{p,z},\ nb_f(e) \qquad \textit{entiers} \tag{2.17}$$

$$\textit{Min} \sum_{e \in E} nb_f(e) \tag{2.18}$$

Ces deux formulations ont des tailles similaires aux programmes linéaires du multiflot : le programme linéaire 3 utilise $\mathcal{O}(|E||Z|+|E|)$ variables et $\mathcal{O}(|V||Z|+|E||Z|+|E|)$ contraintes et le programme linéaire 4 comporte $\mathcal{O}(|\mathcal{P}_Z||Z| + |E|)$ variables et $\mathcal{O}(|\mathcal{P}_Z||Z| + |Z| + |E|)$ contraintes, avec $|\mathcal{P}_Z|$ la taille asymptotique des ensembles $\mathcal{P}_z$.

La résolution du programme linéaire issu du modèle *sommet-arc* est $\mathcal{NP}$-complet à partir de deux commodités [**GJ79**] (Annexe A2, page 216, ND38). Les méthodes de résolution classiques pour un programme linéaire fractionnaire ne s'appliquent plus ici. On a alors recours, en général, à des méthodes de *Branch and Bound/Cut* pour rechercher une solution (cf. [**NW88**] en section II.4.2, page 355).

2.1.3. Extraire des chemins d'un multiflot entier

Le résultat d'un calcul de multiflot avec le modèle *sommet-arc* ne donne pas directement l'ensemble des chemins de routage. Le flot obtenu pour chaque requête doit être décomposé en chemins de longueurs d'onde. Cette décomposition n'est pas unique et peut se faire suivant différents critères. En pratique, compte tenu du problème de groupage décrit en section 1.3.3, nous nous intéressons à la recherche de l'ensemble minimum de chemins réalisant le multiflot obtenu.

La première partie de la figure 2.1 donne un exemple d'écoulement de 11 unités de flot de s à t. Les chemins possibles sont notés $\{X : n_1, \ldots, n_k\}$ avec X la taille du chemin et $n_i,\ i \in \{1, \ldots, k\}$ la suite des nœuds qui forment ce chemin. La décomposition minimale se fait en 3 chemins : $\{(4 : s, A, C, D, t), (5 : s, B, C, E, t), (2 : s, B, C, D, E, t)\}$ comme représenté dans la seconde partie de la figure. Il existe de nombreuses autres décompositions, non

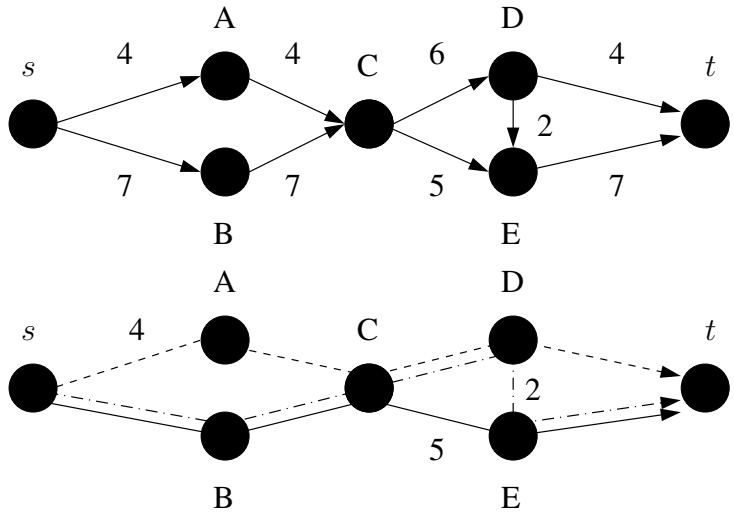


FIG. 2.1 – Flot de s à t sur un exemple de réseau et extraction optimale du nombre minimum de chemins.

minimales, comme par exemple : $\{(4 : s, A, C, E, t), (1 : s, B, C, E, t), (4 : s, B, C, D, t), (2 : s, B, C, D, E, t)\}$.

Proposition 1. *L'extraction du nombre minimum de chemins d'un multiflot entier est un problème $\mathcal{NP}$-complet.*

Démonstration. Une preuve similaire, basée sur le problème de 3-partition a été présentée parallèlement à la rédaction de cette thèse dans [**CCMV04**]. Nous nous contenterons de prouver le théorème sur un réseau simple contenant un nœud de brassage du flot. Ce nœud de brassage reçoit plusieurs parties du flot par différents liens, le flot original étant issu de s. Puis ce nœud doit répartir vers tous ses liens sortants cette quantité de flot. Ainsi, ce nœud brasse le flot entrant vers les liens sortants et notre preuve est basée sur la détermination des chemins qui traversent ce nœud. La preuve s'étend naturellement à tout réseau comportant plusieurs nœuds de brassage. Soit le graphe $G = (V, E)$ représenté en figure 2.2.

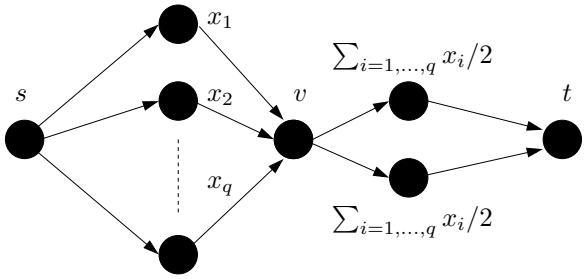


FIG. 2.2 – Représentation d'un réseau avec brassage du flot en v.

La source s produit un flot $S = \sum_{i=1,\cdots,q} x_i$ absorbé par le puits t. Ce flot est subdivisé par q arêtes sortantes de s, chacune acheminant un flot de taille $x_i > 0$ en v. Dans cet exemple, le nœud v ne possède que deux arcs sortant acheminant chacun un flot de $S/2$. Les flots étant tous entiers, on supposera que S est pair, la preuve étant similaire pour S impair.

Comme les x_i sont tous positifs et non nuls, on a au moins q chemins à allouer pour satisfaire ce flot. Dans cet exemple, on peut facilement construire une solution à $q+1$ chemins : on choisit l'ensemble $\{x_1, \cdots, x_k\}$ tel que $\sum_{i=1,\cdots,k} x_i \leq S/2 < \sum_{i=1,\cdots,k+1} x_i$. Si $\sum_{i=1,\cdots,k} x_i = S/2$ on a une décomposition en q chemins du flot. Dans le cas contraire, on coupe le flot x_{k+1} en deux flots de taille $x'_{k+1} = S - \sum_{i=1,\cdots,k} x_i$ et $x''_{k+1} = x_{k+1} - x'_{k+1}$. On crée alors un chemin transportant un flot de taille x'_{k+1} qui complète l'ensemble de k premiers chemins et dont la somme vaut exactement $S/2$. On obtient au final une décomposition en $q+1$ chemins.

La décomposition en $q+1$ chemins étant assurée, il reste à déterminer si elle est possible en q chemins. Ce problème se ramène alors à un problème de partition classique : on doit partitionner l'ensemble $\{x_1, \cdots, x_q\}$ en F et $\overline{F}$ tel que $\sum_{i,\ x_i \in F} x_i = \sum_{i,\ x_i \in \overline{F}} x_i$ qui est un problème $\mathcal{NP}$-complet [**GJ79**] (Annexe A3, page 223, SP12). □

2.2. Heuristiques de génération et d'extraction de chemins

Le modèle *arc-chemin* présenté précédemment utilise des ensembles de chemins candidats pour la construction du programme linéaire. À l'inverse, le modèle *sommet-arc* nécessite d'extraire les chemins du multiflot obtenu, ce qui peut se révéler difficile suivant le critère de qualité d'extraction considéré. Nous présentons dans cette section plusieurs heuristiques de génération de chemins, puis nous proposons une heuristique efficace d'extraction de chemins. À partir de ces méthodes nous mettons en évidence leur efficacité dans la section 2.5, consacrée aux résultats expérimentaux.

2.2.1. Génération de chemins

Avec l'utilisation du modèle *arc-chemin* on doit générer des ensembles de chemins de tailles bornées, pour obtenir un programme linéaire dont la taille est polynomiale. Ces ensembles font référence à l'ensemble des chemins possibles $\mathcal{P}_z$ introduit dans la section 2.1.1. Nous présentons dans cette partie plusieurs heuristiques de générations de chemins. Notons μ un paramètre entier qui contrôle "l'éloignement" des chemins générés par rapport au plus court chemin dont la longueur est $MNH(z)$ (*Minimum Number of Hops*). Le paramètre nb_c contrôle le nombre de chemins générés.

2.2.1.1. *Génération en largeur.* La première heuristique consiste à générer l'ensemble des chemins possibles sans cycle pour une requête $z = (s, t)$ en explorant le graphe en largeur, à partir du nœud s, et en limitant le nombre de chemins générés ainsi que leur taille par rapport au plus court chemin. Nous décrivons tout d'abord le processus de génération de ces chemins, puis nous introduisons les conditions de l'heuristique qui permettent de limiter cet ensemble.

Génération de tous les chemins. Cette méthode est basée sur le processus récursif suivant : on dispose de l'ensemble des chemins $C(k, z)$ générés pour une profondeur k, par rapport à s. On parcourt en largeur la profondeur $k+1$ et on met à jour l'ensemble des chemins de la manière suivante : On retire un chemin $p = (s, u_1, \cdots, u_k)$ de $C(k, z)$. Soit $v_1, \cdots, v_{k'}$ les voisins de profondeurs $k+1$ du nœud u_k. On génère k' chemins à partir de p, $(s, u_1, \cdots, u_k, v_1)$, $\cdots$, $(s, u_1, \cdots, u_k, v_{k'})$, et on ajoute ces chemins dans $C(k+1, z)$. Il se poursuit jusqu'à vider $C(k, z)$. Ce processus est bien sûr exponentiel en mémoire et en temps, s'il n'est pas limité par des critères. Il ne peut donc pas, en l'état, être utilisé tel quel, surtout si le graphe considéré est trop grand.

Critères d'arrêt. L'heuristique de génération en largeur utilise le processus récursif précédent ainsi que certaines règles d'arrêt, ce qui permet alors de limiter l'explosion combinatoire de la méthode. Au rang k de la méthode, pour $p = (s, u_1, \cdots, u_k)$, $\mu \geq 0$, et $v_1, \cdots, v_{k'}$ les voisins de profondeurs $k+1$ du nœud u_k :

- si $v_i = t$, le chemin $(s, u_1, \cdots, u_k, t)$ est trouvé et est ajouté dans $\mathcal{P}_z$.
- si le voisin v_i de u_k appartient déjà à p, une boucle est créée. On rejette donc le chemin engendré par v_i.
- si la longueur (en nombre d'arêtes) de p est supérieure à $MNH(z) + \mu$, le chemin est rejeté.
- si $C(k, z)$ et $C(k+1, z)$ sont vides, la génération est terminée.
- enfin, si $|\mathcal{P}_z| > nb_c$, la génération est terminée.

Exemple de décomposition. La figure 2.3 présente une décomposition sur 3 niveaux pour $\mu = 0$, ce qui stoppe très rapidement l'exploration. Les chemins (s, A, t) et (s, B, t) sont créés.

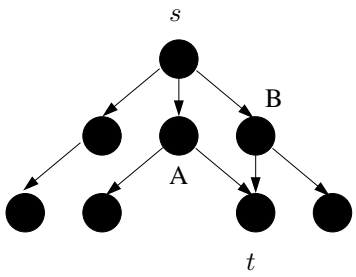


FIG. 2.3 – Exemple de décomposition pour $z = (s, t)$ et $\mu = 0$.

Cette méthode ne permet pas d'avoir des chemins candidats "de bonne qualité", comme le montrent les résultats expérimentaux de la section 2.5.1. À cause du double critère d'arrêt, les chemins générés ne sont pas diversifiés et restent groupés autour du plus court chemin : pour une même taille d'ensemble, la variation de μ n'a que peu d'influence. Nous proposons donc une seconde méthode qui permet de prendre en compte de façon plus significative le critère μ et qui évite de concentrer les chemins autour du plus court.

2.2.1.2. *Marche aléatoire.* La deuxième méthode présentée ici permet d'éviter les inconvénients de la méthode précédente. Cette méthode classique de génération de chemins est similaire à la marche aléatoire proposée par Raghavan que nous présentons ensuite en section 2.3.1. On construit une marche aléatoire débutant au nœud s et tentant de rejoindre le nœud t. Etant donné son caractère aléatoire (et ici, non guidé par des poids) il est probable que la marche n'arrive pas en t. Dans ce cas, le chemin est abandonné au profit d'un nouveau tirage. La construction peut aussi se décrire comme le processus récursif suivant :

Génération de tous les chemins. À l'étape k du processus, un chemin $p = (s, u_1, \cdots, u_k)$ a été généré par le processus de marche aléatoire. Soit $v_1, \cdots, v_{k'}$ les voisins du nœud u_k. On tire avec une probabilité de $1/k'$ le voisin u_{k+1} dans l'ensemble $v_1, \cdots, v_{k'}$. Le nouveau chemin courant est alors $p = (s, u_1, \cdots, u_k, u_{k+1})$.

Critères d'arrêt. Décrivons, comme précédemment, les conditions d'arrêt du processus.

- si $u_{k+1} = t$, le chemin $(s, u_1, \cdots, u_k, t)$ est trouvé et est ajouté dans $\mathcal{P}_z$.

– si le voisin u_{k+1} de u_k appartient déjà à p une boucle est créée. On rejette donc le chemin engendré par le choix de u_{k+1}.
– si la longueur (en nombre d'arêtes) de p est supérieure à $MNH(z) + \mu$, le chemin est rejeté.
– enfin, si $|\mathcal{P}_z| > nb_c$, la génération est terminée.

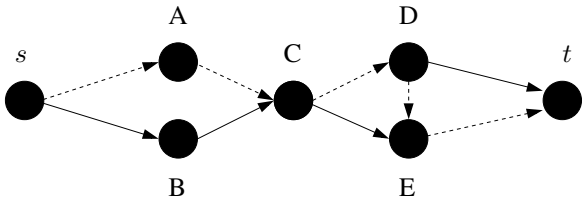


FIG. 2.4 – Marche aléatoire de s à t et $\mu = 1$.

2.2.1.3. *Construction en faisceaux.* Il est peu probable qu'il soit nécessaire de devoir utiliser des chemins trop éloignés des plus courts pour obtenir une bonne solution lors d'un calcul du routage : on souhaite obtenir un compromis entre des chemins très proches du plus court et des chemins longs utilisant des détours. Ainsi, les deux méthodes présentées permettent de construire des chemins en "faisceau" autour du plus court chemin : la génération en largeur limitée par une distance maximum empêche d'avoir des chemins trop éloignés du plus court et la marche aléatoire est aussi contrôlée par une condition de taille maximum. La réduction à un "faisceau" de chemins participe largement à la réduction de la taille de l'instance à résoudre dans le cas du modèle arc-chemins.

Le paramètre μ est très important car il contrôle la largeur du faisceau, comme le montrent les résultats expérimentaux en section 2.5. Plus μ augmente, plus les chemins générés sont longs, consommant ainsi beaucoup plus de capacité sur les câbles si le flot représente le routage de communications sur un réseau. On a donc intérêt à limiter la taille des chemins, par exemple, à deux fois la longueur du plus court chemin. On parle alors d'un *stretch factor* de 2 fixant μ tel que $0 \leq \mu \leq \max_{z \in Z} MNH(z)$. Intuitivement, on ne souhaite pas consommer plus du double de ressources réseaux pour l'acheminement des requêtes (consommation de la protection $1+1$), par rapport au minimum nécessaire obtenu si l'on pouvait router toutes les requêtes sur les plus courts chemins. En pratique, la valeur de μ est déterminée expérimentalement, comme décrit en section 2.5.1.

Il faut noter que cette génération est statique, en comparaison des méthodes de routage de trafic ou la génération des chemins est guidée dynamiquement par le trafic déjà routé. Une telle génération dynamique est utilisée dans la section 2.3.2 pour un algorithme d'approximation du multiflot. Cependant, la génération statique de chemins sera utile pour les modèles de protection du chapitre 4 utilisant ces ensembles de chemins précalculés à l'avance.

2.2.2. Extraction des chemins

L'extraction des chemins, présentée en section 2.1.3, à partir d'un multiflot solution étant un problème $\mathcal{NP}$-complet, nous proposons une heuristique, appelée *Allocation maximum* pour extraire un ensemble de chemins d'une solution.

L'heuristique d'*Allocation maximum* est un algorithme glouton d'allocation de routes ordonnées par taille du flot. L'idée sous-jacente est d'essayer d'extraire chaque route en consommant un maximum de flot, sur chaque arc. Le principe général de l'algorithme repose sur une

Algorithme 1 Algorithme d'*Allocation maximum*.

Entrée: $Z, G = (V, E), \forall e \in E,\ \forall z \in Z,\ x_{e,z}$ le flot calculé
Sortie: Ensemble de routes $\mathcal{R}(z),\ \forall z \in Z$

1: **pour tout** $z = (s, t) \in Z$ **faire**
2: $taille_{totale} = 0$, $chemin = ()$
3: **tant que** $taille_{totale} \neq size(z)$ **faire**
4: $v = s, taille_{cur} = +\infty$
5: {Construction d'un chemin}
6: **tant que** $v \neq t$ **faire**
7: Choisir u dans $E^+ = \{\delta^+(v)\}$ tel que $x_{u,z} \geq x_{e,z},\ \forall e \in E^+$
8: $chemin = concat(chemin, u)$ {Concatène u à $chemin$}
9: $taille_{cur} = \min(taille_{cur}, x_{u,z})$
10: $v = extremite(u, v)$ {Récupère le nœud connectant v par u}
11: **fin tant que**
12: $chemin = concat(chemin, t)$, $taille_{totale} = taille_{totale} + taille_{cur}$
13: Ajouter à $\mathcal{R}(z)$ la route $chemin$ de taille $taille_{cur}$
14: {Mise à jour du flot restant à allouer}
15: **pour tout** $e \in chemin$ **faire**
16: $x_{e,z} = x_{e,z} - taille_{cur}$
17: **fin pour**
18: **fin tant que**
19: **fin pour**

construction itérative d'une route sur le graphe valué par le flot : à chaque nœud on choisit l'arête sortante portant le maximum de flot. Lorsque le chemin est construit, on retranche la valeur minimum du flot rencontrée lors de la construction de la route sur toutes les arêtes de cette route.

A partir d'un graphe $G = (V, E)$ et de la solution du modèle sommet-arc dans la variable $x_{e,z}$, nous décrivons plus formellement l'heuristique d'*Allocation maximum* dans l'algorithme 1. La preuve de la correction de l'algorithme se fait simplement en remarquant qu'à un multiflot entier respectant les lois de Kirchoff, on retire un flot (puisque c'est un chemin) dans la dernière boucle de l'algorithme (ligne 15, 16, 17) respectant lui aussi les lois de Kirchoff. Dans ce cas, le nouveau multiflot obtenu respecte lui aussi les lois de Kirchoff, ce qui permet à l'itération suivante (ligne 3) de poursuivre le processus d'extraction sur un flot valide. En outre, le chemin produit consomme une partie du flot en respectant les contraintes de capacité puisque sa taille est bornée tout au long du processus par le minimum des flots le long du chemin construit (ligne 9).

2.3. Méthodes d'approximation du multiflot

Les difficultés inhérentes à la combinatoire du problème du multiflot entier, exposées en section 2.1.1, nous poussent à utiliser des algorithmes d'approximation pour obtenir des résultats dans des temps raisonnables. Il est assez naturel de relaxer la contrainte d'intégrité des variables pour revenir à des calculs polynomiaux de multiflots. Il faut ensuite utiliser des méthodes permettant d'en extraire un routage entier et par conséquent, le multiflot entier résultant.

Algorithme 2 Arrondi aléatoire par Raghavan [**Rag94**].

Entrée: $G = (V, E),\ Z,\ x_{e,z}$
Sortie: Un ensemble de routes $\mathbb{R}_z,\ \forall z \in Z$

pour tout $z = (s, t) \in Z$ **faire**
  pour tout unité de flot routant z **faire**
    Soit $v = s$. Créer un chemin vide $p = ()$
    {Marche aléatoire de s à t dans G}
    tant que $v \neq t$ **faire**
      Choisir e dans $\delta^+(v)$ avec une probabilité de $x_{e,z}/(\sum_{u \in \delta^+(v)} x_{u,z})$
      $p = concat(p, e)$
      $v = extremite(e, v)$
    fin tant que
    Sauver p dans $\mathcal{P}_z$
  fin pour
fin pour

2.3.1. Arrondi aléatoire du multiflot fractionnaire

Les programmes linéaires précédemment présentés peuvent être résolus polynomialement si l'on relâche la contrainte d'intégrité. Par la suite, nous présentons dans la section 2.3.2 une approximation du multiflot fractionnaire. Dans cette partie, nous montrons comment déduire d'un multiflot fractionnaire une solution entière pour répondre au problème du routage en utilisant l'algorithme de Raghavan [**RT87, Rag94**] fondé sur des techniques d'arrondis aléatoires. Cette présentation introduira une version modifiée de cette algorithme ainsi qu'une preuve de correction qui sont présentés en section 4.5.2 dans un algorithme dédié à la protection.

L'algorithme de Raghavan dans [**RT87**] est fondé sur l'idée que l'on sait calculer une solution optimale fractionnaire d'un programme linéaire en un temps raisonnable avec des méthodes polynomiales. En partant du résultat de tout programme linéaire, on peut chercher à arrondir les variables du programme pour les rendre entières. Ce faisant, on introduit trois problèmes :

- On risque de dénaturer la cohérence de la solution, par exemple en ne respectant plus les lois de Kirchoff.
- On risque de s'éloigner de la valeur de la solution fractionnaire tout en ne garantissant aucunement une convergence vers la solution optimale entière.
- Les contraintes de majoration (ou de minoration) du programme linéaire peuvent ne plus être respectées puisque l'on modifie les valeurs des variables et qu'en les augmentant, on fait aussi augmenter les sommes linéaires de ces variables.

L'idée de Raghavan est de donner une méthodologie pour arrondir les variables du programme linéaire. Cette méthodologie a pour but de :

- Garantir que certaines contraintes continuent d'être respectées.
- Quantifier la qualité du flot obtenu, par rapport à l'optimal.
- Garantir que la probabilité de violer une contrainte est faible.

On utilise une marche aléatoire pour parcourir le multiflot : chaque marche aléatoire permet de créer une route d'une seule unité de flot sur le graphe. L'idée majeure de l'algorithme est d'interpréter les flots fractionnaires des arcs comme la probabilité pour un chemin de choisir cet arc. La marche, contrairement à celle de la section 2.2.1.2 n'est pas équiprobable au niveau des nœuds. Elle est conduite par la valeur des flots sur les arcs du graphe.

L'algorithme 2 présente une version détaillée de l'algorithme dans son ensemble. Notons que la méthode *extremite(e,v)* donne u si $e = (v, u)$. La marche aléatoire s'effectue au niveau de la boucle allant de la ligne 5 à la ligne 9. La boucle extérieure ne parcourt que les requêtes, en arrondissant chaque unité de flot indépendamment. Si l'on souhaite reconstituer des chemins routant plusieurs unités de flot, il suffit d'agréger les routes unitaires identiques. L'algorithme total est bien entendu polynomial : sa complexité est de l'ordre de $\mathcal{O}(|Z|.m)$ où $|Z|$ doit être considéré comme le nombre total de requêtes unitaires.

Les détails de la qualité de la solution obtenue sont donnés en section 4.5.2, pour une version modifiée de cet algorithme. Le flot total ainsi arrondi est d'une valeur d'au moins $F(1-\varepsilon^2)$ avec probabilité $2\exp(-0,38\varepsilon^2 F)$ où F est le flot entier optimal et ε un paramètre qui contrôle le compromis entre une valeur du flot proche de F et une bonne probabilité. Concernant la probabilité de violation des contraintes de capacité, pour $c(e)$ assez grand ($c(e) \geq 12 \ln |E|$), les contraintes sont respectées avec probabilité $1 - \frac{1}{|E|}$.

2.3.2. Approximation du multiflot fractionnaire

La section précédente introduit une méthode pour extraire d'un multiflot fractionnaire un multiflot entier. Nous avons supposé qu'un tel multiflot fractionnaire pouvait être obtenu par des méthodes du type programmation linéaire : nous introduisons dans cette partie une approximation du multiflot fractionnaire qui permet d'obtenir une solution approchée de manière plus efficace.

Nous présentons l'algorithme de Fleisher [**Fle00**], issu lui même des travaux de Garg et Konemann [**GK98**]. L'algorithme de Fleisher repose sur l'interprétation du dual du programme linéaire présenté en section 2.1.1 et calcule un multiflot fractionnaire maximum sous des contraintes de capacité. Là ou le primal cherche à maximiser une expression linéaire sous des contraintes convexes majorantes, le dual minimise une autre expression linéaire (une combinaison linéaire de l'objectif du primal) sous des contraintes convexes minorantes. Plus formellement, pour un primal de la forme $\max\{c^T x |\ Ax \leq b,\ x \in \mathbb{R}^n\}$, on peut, de manière équivalente, travailler sur un dual de la forme $\min\{b^T y |\ A^T y \geq c,\ y \in \mathbb{R}^{+\,m}\}$ si $A \in \mathbb{R}^{m \times n}$.

Intuitivement, le dual du programme linéaire du multiflot fractionnaire s'interprète comme la construction d'une métrique l sur les arcs du graphe telle que le poids d'un arc correspond à la quantité de flot qui le traverse. A partir de cette métrique, on réalise des poussées successives de flot sur le plus court chemin, celui de poids minimum avec la métrique l. Ainsi, on réalise une génération dynamique de chemins, à partir du flot déjà alloué, par des calculs successifs de plus courts chemins.

L'algorithme 3 présente une version détaillée de l'approximation : il construit un flot maximum valide en ne considérant que la métrique associée au dual. Les notations suivantes sont utilisées : SSSP est un algorithme de calcul de plus court chemin à partir d'une source unique (*Single Source Shortest Paths*), $l(e)$ est la longueur d'une arête et $l(chemin)$ celle d'un chemin, c'est-à-dire tel que $\sum_{e \in chemin} l(e) = l(chemin)$. Le poids initial d'un arc est une constante $\delta > 0$, correspondant à une quantité de flot nulle (ligne 1). Ensuite, l'algorithme procède à des poussées de flots successives le long de plus courts chemins (SSSP) pour chaque commodité (lignes $3 - 11$). Pour une requête de s à t, on pousse sur le plus court chemin $chemin$ de capacité minimale c_m la quantité de flot $c_m/(\log_{1+\epsilon} \frac{1+\epsilon}{\delta})$, puis pour chaque arc $e \in chemin$ on augmente $l(e)$ d'un facteur $1+\epsilon\frac{c_m}{c(e)}$ (lignes $5-10$). Cette poussée de flot sur un chemin ayant un poids calculé par une métrique particulière s'interprète comme la génération dynamique d'un chemin qui évite les arcs engorgés et donc ayant un poids proche de 1. L'algorithme termine lorsque tous les plus courts chemins ont une longueur supérieure à 1.

Algorithme 3 $(1+\epsilon)$-approximation du multiflot maximum par Fleischer [**Fle00**].

Entrée: $G=(V,E),\ |V|=n,\ |E|=m,\ Z,\ \epsilon>0$
Sortie: l métrique telle que tout chemin $s \to t$ est de longueur > 1.
Sortie: f $(1+\epsilon)$-approximation du max-multiflot de Z dans G

1: {Initialisation} $\forall e \in E, l(e) = \delta = (1+\epsilon)((1+\epsilon)n)^{-\frac{1}{\epsilon}}, x_{e,z} = 0$
2: {Borne sur la longueur d'un SSSP } $\lambda = \delta$
3: **tant que** $\lambda \leq 1+\epsilon$ **faire**
4: **pour tout** $z=(s,t) \in Z$ **faire**
5: $chemin \leftarrow$SSSP $(s \to t)$
6: **tant que** $l(chemin) \leq (1+\epsilon)\lambda$ **faire**
7: $c_m \leftarrow \min_{e \in chemin} c(e)$
8: $\forall e \in chemin,\ x_{e,z} \leftarrow x_{e,z} + c_m/(\log_{1+\epsilon} \frac{1+\epsilon}{\delta})$
9: $\forall e \in chemin,\ l(e) \leftarrow l(e)(1+\epsilon\frac{c_m}{c(e)})$
10: $chemin \leftarrow$SSSP $(s \to t)$
11: **fin tant que**
12: **fin pour**
13: $\lambda \leftarrow \lambda(1+\epsilon)$
14: **fin tant que**

L'argument principal de l'analyse de complexité, donné dans [**GK98**] et adapté pour cet algorithme dans [**Fle00**], est que, au pire, à chaque poussée de flot le plus court chemin ne contient qu'un seul arc et sa longueur augmente d'un facteur $1+\epsilon$, car $c_m = c(e)$. La longueur initiale des arcs est δ et l'algorithme termine au pire lorsque tous les arcs ont longueur $1+\epsilon$. Donc il y a, au pire, $m \log_{1+\epsilon}\left(\frac{1+\epsilon}{\delta}\right)$ poussées. Vu le choix de δ, imposé pour garantir le facteur d'approximation, cela donne $O\left(\frac{m \log_{1+\epsilon} n}{\epsilon^2}\right)$ poussées, chacune correspondant au calcul d'un plus court chemin dans un graphe où tous les arcs sont de poids positifs (SSSP > 0) de coût $O(m + n\log n)$ avec Dijkstra. Donc, la complexité dans le cas général est $O\left(\frac{m \log_{1+\epsilon} n}{\epsilon^2}(m + n\log n)\right)$. Un calcul plus explicite de la complexité ainsi qu'une preuve exhaustive de correction de l'algorithme peut être trouvée dans [**Riv03**], section 5.2, page 129.

2.3.3. Amélioration par les plus courts chemins incrémentaux

Dans l'algorithme 3, la boucle *tant que* (lignes $6-10$) effectue plusieurs calculs successifs de SSSP pour une même requête z et, entre deux de ces calculs, seule la longueur l des arcs du plus court chemin de s à t est augmentée. Aussi, on peut se contenter, ligne 10, de *mettre à jour* l'arborescence des plus courts chemins. En effet, l'augmentation de la longueur de certains arcs n'aura d'influence que sur une partie de l'arborescence. Nous introduisons donc un algorithme de mise à jour de l'arborescence des plus courts chemins lors d'une modification des poids d'un chemin de bout en bout[2]. Les algorithmes classiques de mise à jour d'arborescence des plus courts chemins ne prennent en compte que des modifications d'un arc à la fois. Dans la suite, nous généralisons l'algorithme de Frigioni, Marchetti-Spaccamela, et Nanni [**FMSN00**] au cas de la modification de toutes les arêtes sur un chemin de l'arborescence.

[2]Ce travail a fait l'objet des publications [**BCLR03, BCL$^+$03**].

Algorithme 4 Marquage des sommets pour la mise à jour de l'arbre des plus courts chemins.

Entrée: $T(s)$ un arbre des plus courts chemins issus de s, $s \to t$ un plus court chemin.
Sortie: Une bonne coloration des sommets rouges.

1: {Initialisation} $\forall y \neq s$ un sommet de $s \to t$, Enfiler(M,$\langle y, l(s,y)\rangle$).
2: **tant que** Non-Vide(M) **faire**
3: $\langle v, l(s,v)\rangle \leftarrow$ Extraire-Min(M).
4: **si** $\exists\, q \notin M$ voisin non rouge de v t.q. $l(s,q) + l(q,v) = l(s,v)$ **alors**
5: $P(v) \leftarrow q$ {v est rose}
6: **sinon**
7: $couleur(v) \leftarrow$ rouge
8: **pour tout** v fils de $v \notin M$ **faire**
9: Enfiler(M,$\langle v, l(s,v)\rangle$)
10: **fin pour**
11: **fin si**
12: **fin tant que**

La mise à jour de l'arbre des plus courts chemins lors de l'augmentation de la longueur d'un arc se décompose en deux étapes. L'étape 1 colorie l'ensemble des sommets V du graphe comme suit :

– $q \in V$ est **rouge** si sa distance à s augmente.
– $q \in V$ est **rose** si q préserve sa distance à s mais doit changer de père dans $T(s)$.
– $q \in V$ est **blanc** si q ne change ni de distance ni de père.

Cette classification permet de passer à l'étape 2 qui calcule les nouvelles distances pour les sommets *rouges* en appliquant une procédure similaire à l'algorithme de Dijkstra. On évite ainsi des calculs inutiles sur l'arborescence non rouge lorsqu'il s'agit de mettre à jour l'arborescence des plus courts chemins. Dans l'algorithme 4 nous détaillons le processus de coloration utilisé : la distance de s à un nœud u est notée $l(s,u)$ et l'arbre $T(s)$ des plus courts chemins issus de s est implicitement décrit par l'ensemble des pères des nœuds dans $T(s)$, notés $P(v),\ \forall v \in V$.

L'algorithme commence par insérer les sommets du chemin $s \to t$ dans une queue M (ligne 1). Cette queue ne contiendra que les sommets touchés par la modification (les rouges et les roses). Ensuite, tant que M n'est pas vide (ligne 2), il traite le sommet le plus proche de s (ligne 3) et vérifie si sa distance est conservée (ligne 4). Dans ce cas, son père est mis à jour, le sommet est implicitement marqué rose (ligne 5), et tout le sous-arbre issu de ce sommet est implicitement marqué blanc vu qu'il ne sera jamais inséré dans M. Dans le cas contraire, le sommet est marqué rouge (ligne 7) et ses fils qui ne le sont pas déjà sont insérés dans M puisqu'ils doivent au moins être roses (lignes 8, 9).

La preuve de la validité de cet algorithme de coloration est donné ci après :

Lemme 1 (validité du coloriage)**.** *Soient $G = (V, E)$ un graphe, $T(s)$ l'arborescence des plus courts chemins enracinée en s et $P_{s,t}$ le plus court chemin de s à t. Si l'on augmente la longueur de tous les arcs du chemin $P_{s,t}$ alors l'algorithme 4 colorie correctement tous les sommets de G.*

Démonstration. La validité du coloriage découle des propriétés suivantes.

($\mathcal{P}_1$) Tous les sommets de $P_{s,t}$ à l'exception de la racine s sont insérés dans M.

($\mathcal{P}_2$) Tous les fils d'un sommet rouge sont insérés dans M. Tous les fils qui ne sont pas sur le chemin $P_{s,t}$ d'un sommet rose ne sont jamais insérés dans M.

($\mathcal{P}_3$) Un sommet est blanc si et seulement si il n'a jamais été inséré dans M.

($\mathcal{P}_4$) Les sommets sont extraits de M par ordre de distance croissante : si u et q sont présents dans M, pas nécessairement simultanément, alors $d(u) < d(q)$ implique que u est extrait avant q.

Les propriétés ($\mathcal{P}_1$) et ($\mathcal{P}_2$) sont des conséquences triviales de l'algorithme 4 (ligne 1 pour ($\mathcal{P}_1$), ligne 4 à 9 pour ($\mathcal{P}_2$)). ($\mathcal{P}_3$) est due au fait que l'on n'enfile que les fils des sommets rouges qui sont, par définition, rouges ou roses.

Reste à prouver ($\mathcal{P}_4$). On note $t(x)$ la "*date*" à laquelle le sommet x est extrait de M. ($\mathcal{P}_4$) est trivialement vraie lorsque le premier sommet est extrait.

Soit v un sommet. Supposons, par récurrence, que ($\mathcal{P}_4$) est vraie pour tout $u, u' \in \Delta(v)$, l'ensemble des sommets extraits avant v. Au temps $t(v)$, $v = \min\{x \in M\}$. Soit $P(v) \in \Delta(v)$ le père de v.

- $\forall u \in \Delta(v)$ t.q. $t(u) < t(P(v))$, ($\mathcal{P}_4$) et le fait que le père de v ait une distance à s inférieure à celle de v impliquent que $D(u) \leq D(P(v)) \leq D(v)$.
- $\forall u \in \Delta(v)$ t.q. $t(P(v)) < t(u)$, par définition de $\Delta(v)$ $t(u) < t(v)$. De plus, puisque v est inséré dans M au plus tard quand $P(v)$ en est extrait (lignes 8 et 9), il en suit que v est déjà dans M à $t(u)$. Ainsi, u et v sont dans M simultanément à $t(u)$. M étant une file, à $t(u)$ $u = \min\{x \in M\}$, et donc $D(u) < D(v)$.

Conséquemment, ($\mathcal{P}_4$) est vraie pour $\Delta(v) \cup \{v\}$.

□

Notons qu'il est difficile d'exprimer la complexité de cet algorithme de mise à jour, appelé USSSP (*Updated Single Source Shortest Path*) pour la raison suivante : dans [**FMSN00**], les auteurs l'expriment en fonction du nombre α de sommets changeant soit de père soit de distance, et d'une constante β dépendant de la classe de graphes considérée ($\beta \leq 3$ pour les graphes planaires, $\beta \leq d$ pour les graphes de degré maximum d, ..., $\beta = \mathcal{O}(\sqrt{m})$ pour les graphes généraux). La complexité d'USSSP est alors $\mathcal{O}(\alpha\beta \log n)$ et est toujours inférieure à la complexité de l'algorithme de Dijkstra, $\mathcal{O}(m + n \log n)$. La complexité de l'algorithme de Fleisher ainsi modifié devient alors :

$$\frac{\log_{1+\epsilon} n}{\epsilon^2} \left(|Z|.\text{SSSP} + (m - |Z|).\text{USSSP}\right).$$

2.4. Routage et affectation de longueurs d'onde

L'algorithme d'approximation du multiflot fractionnaire permet de router un multiflot sur un réseau modélisé par un graphe dont les arcs sont valués par leur capacité. Ce modèle ne prend pas en compte le brassage des longueurs d'onde au niveau des nœuds : [**CR02**] introduit ce brassage en proposant de dupliquer le graphe représentant le réseau en autant de couches que la multiplicité en longueurs d'onde, W. Ainsi, chaque couche du réseau représente une longueur d'onde donnée sur le réseau et le passage au travers d'un nœud de cette couche se traduit par une continuité optique du signal. Dans ce cas précis, le routage d'un multiflot sur un tel modèle revient à résoudre un problème d'affectation de longueurs d'onde, du type RWA, présenté en section 1.3.2.

La figure 2.5 présente la transformation du graphe initial (graphe contenu dans le plan gris) vers le graphe en couche ou graphe auxiliaire. Des sommets supplémentaires sont ajoutés : pour chaque nœud source s du réseau initial, on crée un super nœud source qui relie les duplicata de s (un par couche). On fait de même pour chaque nœud destination t, on crée un super nœud

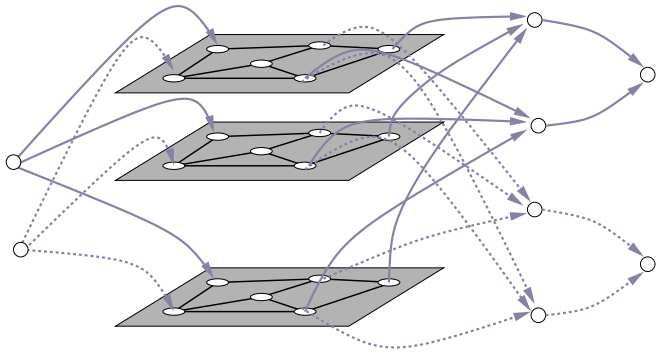


FIG. 2.5 – Exemple de graphe en couche, pour 4 requêtes issues de 2 sources et $W = 3$.

destination. Enfin, on ajoute un nœud destination particulier qui servira de destination pour toutes les requêtes allant en t (on relie alors chaque super nœud destination vers ce nouveau nœud). On peut se référer à [**CR02**] pour les détails de la construction.

Notons que ce modèle ne prend pas en compte la possibilité de convertir des longueurs d'onde au niveau des brasseurs optiques. L'extension de ce modèle est présenté dans [**CR02**] : on ajoute un certain nombre d'arcs entre plusieurs copies d'un même nœud ce qui permet de changer de couche dans le graphe auxiliaire. On réalise ainsi explicitement une conversion d'une longueur d'onde λ_1 en une autre longueur d'onde λ_2. Dans la version de la figure 2.5, il n'y a pas de possibilité de conversion de longueur d'onde : il n'y a donc aucun lien entre deux couches (grises sur la figure 2.5).

Si le réseau WDM a n nœuds et m liens, lorsqu'il y a W longueurs d'onde et une instance de communication qui contient au moins toutes les paires possibles (le pire des cas), le graphe auxiliaire a $O(n^2)$ sommets et $O(W.n^2)$ arcs. Si l'on ne regarde pas plus loin, la complexité de l'algorithme de Fleischer est donc :

$$O\left(\frac{\log_{1+\epsilon} n}{\epsilon^2} W n^4 (W + 2\log n)\right).$$

Cependant, remarquons que la structure du graphe amène à une spécialisation simple de l'algorithme de plus courts chemins : nous ne nous intéressons pas à un arbre des plus courts chemins, mais plutôt à un chemin dans cet arbre, allant d'une source à une destination. Ce chemin traverse une seule et unique couche du graphe auxiliaire, ce qui limite le nombre de voisins touchés par l'algorithme USSSP : il suffit de faire la mise à jour de la structure dans la couche où la métrique a changé puis de mettre à jour les liaisons aux extrémités, ce qui se fait en temps $O(\log nW)$ en utilisant un tas. La complexité totale d'USSSP devient $O(\alpha\beta \log nW) < O(m+n\log n)$. Il est important de noter que dans ce cas, on peut minorer le gain de l'utilisation d'USSSP par un facteur W.

De plus, lors d'un calcul de plus court chemin entre un super nœud source et un super nœud destination, les autres super nœuds sources et super nœuds destinations peuvent être oubliés dans l'algorithme de mise à jour. Ainsi, le nombre de sommets considérés à chaque calcul de plus courts chemins n'est pas en $O(n^2)$ mais en $O(nW)$ et le nombre d'arcs est en $O(mW)$. Toutefois, le nombre d'arcs considérés dans l'algorithme de multiflot reste en $O(Wn^2)$, soit la totalité du graphe. En conséquence, un calcul de multiflot sans conversion, l'algorithme de

Fleischer prend un temps au pire

$$O\left(\frac{\log_{1+\epsilon} n}{\epsilon^2} n^2 W(m + n \log n)\right)$$

soit une amélioration d'un facteur $O(W)$ par rapport à la précédente version et d'un facteur $O\left(\frac{n^4 W^2 \epsilon^2}{\log n}\right)$ par rapport au $O(n^8 W^3)$ nécessaire à la résolution du programme linéaire présenté dans [**CR02**].

L'algorithme USSSP se comporte d'autant mieux que la topologie des réseaux présentés en annexe A.2 n'est pas régulière. Le nombre de nœuds est assez réduit et le diamètre des réseaux est faible. Lors de la détermination d'un plus court chemin, le nombre d'arcs touchés reste limité, réduisant d'autant plus le temps passé dans USSSP.

2.5. Résultats expérimentaux

Nos expérimentations utilisent les réseaux détaillés en annexe A.2. Toutes nos expérimentations ont été effectuées sur des PC standards du type PENTIUM™ IV cadencés à 2Ghz avec 1Go de mémoire.

Selon les algorithmes, l'occupation mémoire ou la vitesse d'exécution sont des critères plus ou moins critiques pour attaquer des instances réelles. Par exemple, lors de la génération des chemins dans $\mathcal{P}(z)$ à l'aide de la méthode de *Génération en largeur*, l'espace mémoire peut très vite être saturé alors que la génération de ces chemins à l'aide d'une marche aléatoire ne nécessite pas une puissance de calcul énorme. Le phénomène inverse s'observe sur les algorithmes d'approximation du multiflot qui nécessitent un temps de calcul non négligeable, mais peu de mémoire.

2.5.1. Génération de chemins

Nous présentons en figure 2.6 une comparaison des deux heuristiques de génération de chemins[3]. L'ensemble des chemins sont utilisés au chapitre 4 pour la résolution d'un multiflot de protection mixte, c'est-à-dire combinant le modèle *sommet-arc* et le modèle *arc-chemin*.

Cette expérimentation mesure le multiflot fractionnaire alloué sur le réseau R_1, sans extraction d'une solution entière, lorsque l'on résout le modèle avec CPLEX SOLVER™. Le paramètre μ contrôle l'étalement en faisceau des chemins produits pour les deux algorithmes et c'est la variable dont nous proposons de déterminer la valeur de manière expérimentale.

La méthode utilisant la *marche aléatoire* produit de meilleurs résultats et un résultat "optimal" pour une valeur $\mu = 3$. Pour des valeurs de μ trop faibles, le multiflot n'a aucune flexibilité pour répartir le flot au travers de multiples chemins et il risque même de ne trouver aucune solution à cause de la saturation de certains arcs. Cette saturation a lieu pour l'algorithme de *génération en largeur* qui ne génère pas un nombre de chemins suffisamment disjoints pour éviter des saturations. En principe, pour $\mu = 0$, les deux méthodes devraient produire les mêmes ensembles de chemins, c'est-à-dire les plus courts chemins. Ceci n'est pas le cas à cause de la condition $|\mathcal{P}_z| > nb_c$ qui limite le nombre de chemins générés. Etant donné la topologie de R_1, il y a pour chaque requête plusieurs plus courts chemins. Si on limite le nombre total de chemins générés, on obtient les premiers plus courts chemins (en terme de distance par rapport au

[3] le programme linéaire utilisé est basé sur l'algorithme *BBG*, qui est un flot entier calculant des chemins de protection. Ce modèle est présenté en détail au chapitre 4.

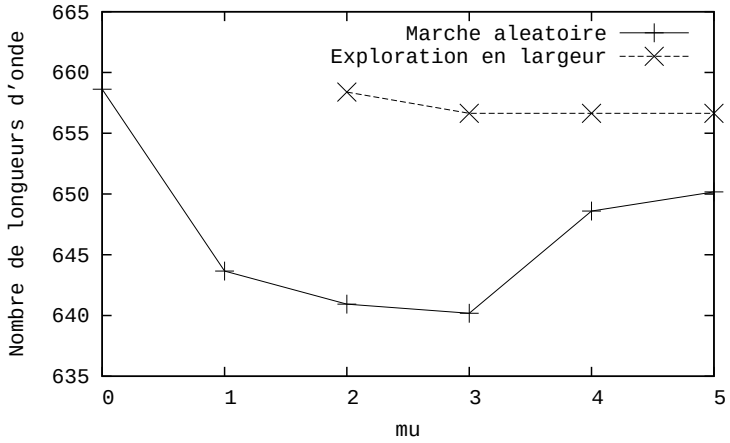


FIG. 2.6 – Nombre total de longueurs d'onde allouées dans R_1 par *BBG*

plus court) parmi l'ensemble des possibilités ; avec la *marche aléatoire*, avec une même limite, on obtient des chemins plus aléatoirement répartis générant un ensemble plus diversifié.

Lorsque μ augmente trop, les chemins s'éloignent du faisceau des plus courts chemins. L'ensemble des chemins est alors beaucoup plus disparate et leur longueur moyenne augmente significativement. Dans ce cas, pour un nombre de chemins générés qui ne varie pas, la qualité de la solution se dégrade, expliquant l'augmentation du nombre de longueurs d'onde à partir de $\mu = 4$.

2.5.2. Routage des requêtes par un multiflot entier

Les résultats calculés à l'aide du programme linéaire 3 sont présentés en section 6.5 dans le chapitre 6 consacré à PORTO. Ces résultats comparent ce routage avec un monoflot et un double monoflot, développés dans le chapitre traitant des problèmes de protection, en section 4.2.2.

2.5.3. Approximation du multiflot fractionnaire

La figure 2.7 montre que notre implémentation de l'algorithme 3 sur le réseau EU permet de traiter des réseaux de grande taille, à l'inverse du solveur de programme linéaire CPLEX SOLVER™. En particulier, il nous devient possible de traiter des réseaux WDM *denses* utilisant plusieurs centaines de longueurs d'onde par fibre alors que CPLEX SOLVER™ échoue sur le réseau EU à partir de ce seuil critique : 13 180 secondes sont nécessaires pour la dernière instance admissible pour CPLEX SOLVER™ comprenant 99 longueurs d'onde. Au delà, la taille mémoire du programme linéaire dépasse les capacités de la machine.

Notons que notre implémentation met en œuvre l'algorithme de Belman-Ford. En effet, la comparaison expérimentale de Belman-Ford et de Dijkstra montre que Belman-Ford est plus efficace pour des graphes de moins de quelques milliers de sommets. Étant donné le nombre d'appels à des calculs de plus courts chemins, il est indispensable d'utiliser l'algorithme le plus performant pour la résolution de nos problèmes sur nos réseaux qui ne possèdent jamais plus d'une centaine de nœuds. L'algorithme de mise à jour est lui aussi basé sur Belman-Ford.

L'expérimentation de la figure 2.8 concerne le réseau NSFNET. Pour que la taille du problème soit conséquente les requêtes ont été multipliées par 20 et le nombre de fibres par lien

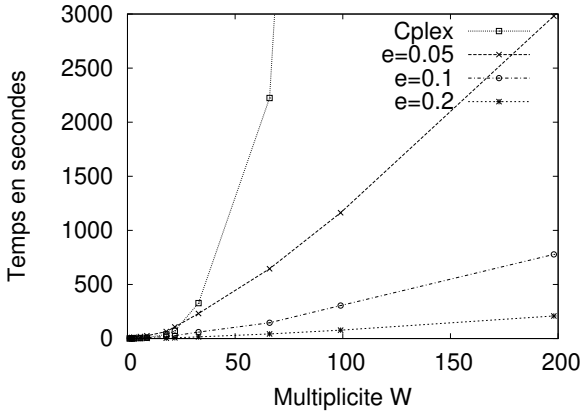


FIG. 2.7 – Temps de calcul pour différents ϵ sur EU.

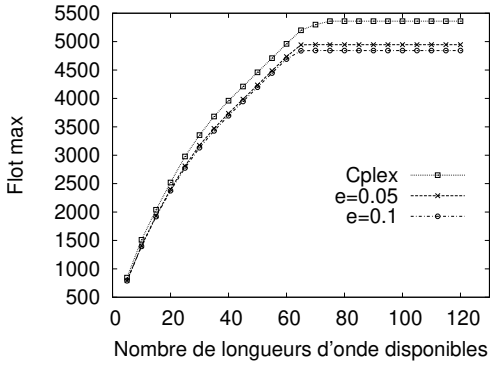


FIG. 2.8 – Valeur du flot maximum comparée à la solution exacte donnée par CPLEX en fonction du nombre de longueurs d'onde sur NSFNET.

a été arbitrairement fixé à 5. La figure 2.8 présente la valeur du flot maximal obtenu par l'algorithme 3 pour $\epsilon = 0.05$ et $\epsilon = 0.1$. Lorsque $\epsilon = 0.1$, la valeur du flot est bien à 10% du flot maximal calculé par CPLEX SOLVER™ (différence de 518 unités de flot). Par contre, pour $\epsilon = 0.05$, la valeur du flot devrait se situer à moins de 5% de la valeur optimale, ce qui n'est pas le cas. En effet, lorsque ϵ est trop proche de zéro, des problèmes de précision numérique apparaissent. Ceux-ci, déjà évoqués dans [**Fle00**], sont dus à la manipulation simultanée d'ordres de valeurs de grandeur très différents.

La figure 2.9 présente le temps d'exécution pour le calcul du flot maximal pour l'instance I du réseau NSFNET, celle-ci étant multipliée par un facteur k. Cette figure illustre bien la dépendance linéaire entre le temps de calcul et le nombre de longueurs d'onde disponibles.

Conclusion

Nous avons présenté les modèles classiques permettant de calculer des multiflots : le modèle *sommet-arc* et *arc-chemin*. Nous avons montré comment les utiliser avec notamment une heuristique permettant de générer des chemins candidats à l'aide d'une marche aléatoire. La

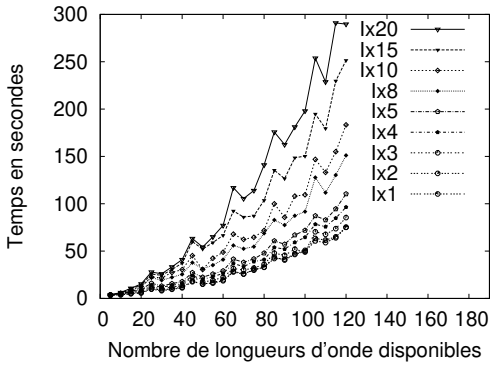


FIG. 2.9 – Temps de calcul pour $\epsilon = 0.05$ sur NSFNET et son instance I, multipliée par $k = 1 \ldots 20$.

résolution de ces modèles en programmation linéaire entière étant $\mathcal{NP}$-complets, nous avons introduit un algorithme d'approximation du multiflot fractionnaire. A partir de la résolution fractionnaire d'un multiflot on obtient, à l'aide d'un technique d'arrondis aléatoires une solution entière, proche de la solution optimale. Nous avons montré aussi les améliorations qui peuvent être apportées à l'approximation du multiflot et les expérimentations qui valident ces améliorations. Enfin, l'approximation est adaptée aux réseaux optiques sans conversion, la spécificité du modèle permettant encore d'améliorer les performances théoriques.

Le chapitre qui suit présente le problème du groupage. Ce problème étant fortement relié au routage dans les réseaux optiques, certains algorithmes de routage pour le groupage sont explicités dans ce prochain chapitre.

CHAPITRE 3

Algorithmes de groupage

Le problème du groupage (*grooming*) dans les réseaux optiques est un thème récent et lié à la multiplication des niveaux d'encapsulation de trafic. Les équipements opto-électroniques sont donc de plus en plus complexes, que ce soit du point de vue de l'interface entre le monde optique et le monde électronique, ou du point de vue de la gestion des communications optiques proprement dites (introduction des concepts de bandes de longueurs d'onde).

Le chapitre 2 a présenté les problématiques du routage des communications dans ces réseaux optiques, l'une des facettes de l'allocation de ressources du réseau. Alors que le routage vise à optimiser les ressources pour l'allocation des câbles, nous proposons dans ce chapitre d'aborder l'allocation des ressources qui concernent les nœuds de ces réseaux.

Le groupage permet de regrouper les conteneurs du trafic dans des conteneurs de plus haut niveau ce qui permet d'économiser des équipements au niveau des nœuds traversés par des routes. L'objectif de ce chapitre est de présenter les modèles dédiés aux réseaux optiques ainsi que des modèles génériques de groupage de conteneurs. Nous détaillons les algorithmes et heuristiques classiques traitant cette problématique. Nous étudions par la suite un algorithme glouton de groupage qui nous servira à introduire une analyse de l'influence du routage sur le groupage.

3.1. Le groupage

Le problème du groupage, présenté en section 1.3.3, peut différer légèrement suivant les types de conteneurs envisagés et les objectifs de ce groupage. On distingue, dans cette section le modèle générique du groupage du modèle propres aux réseaux WDM. La section 1.3.3 parle de l'encapsulation des longueurs d'onde dans des bandes, et des bandes dans des fibres. Cependant, la majorité des méthodes de résolution présentées dans cette partie peuvent s'appliquer (ou s'adapter) à d'autres problèmes d'encapsulation de conteneurs [**MM04**].

3.1.1. Le groupage dans les réseaux WDM

L'objectif du groupage est de déterminer l'équipement au niveau des nœuds du réseau ainsi que le regroupement des bandes dans les fibres et des longueurs d'onde dans les bandes. Le critère utilisé pour quantifier le groupage obtenu est un coût d'exploitation du réseau, qui dépend directement des équipements nécessaires au niveau des nœuds pour gérer les flux optiques, à savoir la taille en nombre de ports des W-OXC, B-OXC et F-OXC.

Ce coût n'a pas forcément une expression analytique simple à utiliser et la plupart des modèles préfèrent supposer que le coût est directement proportionnel au nombre d'entrées et de

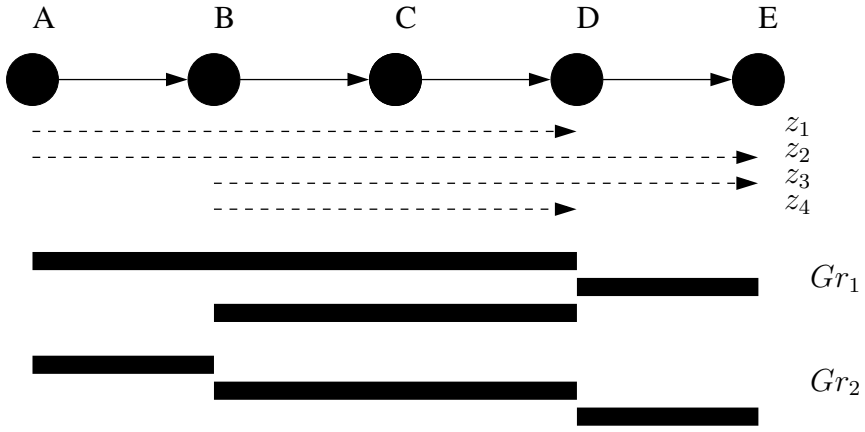


FIG. 3.1 – Comparaison de deux groupages sur un niveau

sorties des équipements des différentes couches. Ainsi dans [**CRD04**] on cherche à minimiser le nombre maximum d'entrées et de sorties sur l'ensemble des nœuds du réseau (objectif "min-max"). L'intérêt de ce choix de modélisation réside dans le fait que l'opérateur déploie le même type d'équipement sur tous les nœuds du réseau et qu'il a donc besoin de connaître (et de réduire) la taille de l'équipement maximum puisque c'est celui-ci qui servira d'équipement étalon.

Dans d'autre études comme [**ZM02**], on revient à l'objectif classique du routage de la section 1.4.2 qui consiste à minimiser l'occupation du réseau ou à maximiser le nombre de connexions acceptées sur le réseau. Cet objectif n'a de sens que si l'on est limité en terme de conversion de longueurs d'onde : si la capacité de conversion est infinie, on peut utiliser optimalement les longueurs d'onde des câbles. Dans certains cas, l'objectif peut même être de limiter le nombre de conversion de longueurs d'onde dans les équipements des nœuds comme dans [**GT03**] où l'on utilise un modèle de programmation linéaire pour réarranger les longueurs d'onde déjà affectées dans les câbles.

Enfin, le nombre de longueurs d'onde multiplexées dans une fibre pouvant aller jusqu'à plusieurs centaines, l'impact du coût total de l'équipement des nœuds devient important : il faut trouver un compromis entre le nombre de longueurs d'onde utilisées et la taille des équipements du type OADM (*Optical Add Drop Multiplexer*) requis [**GRS00, ZQ00**].

3.1.2. Le modèle générique du groupage

À partir de la description du problème du groupage sur les réseaux optiques à 3 niveaux, on peut dégager la problématique suivante : si l'on suppose que l'on possède différents niveaux de conteneurs, comment réaliser des chemins de conteneurs acheminant un routage donné en essayant de grouper dans les conteneurs de plus haut niveau un maximum de trafic, en respectant la contrainte de capacité ?

L'exemple de la figure 3.1 permet d'illustrer les différences entre deux groupages donnés. On a un réseau avec une topologie de chemin de 5 nœuds. La capacité de chaque lien est de 2 conteneurs et un conteneur peut contenir 2 requêtes au maximum. Les requêtes sont représentés en pointillés et sont groupées avec deux ensembles de suite de conteneurs : Gr_1 et Gr_2. On a par exemple la requête z_1 qui utilise, dans le cas du groupage Gr_1 la suite de conteneurs

(A, B, C, D) pour être acheminée. Dans le cas du groupage Gr_2 cette même requête utilise le conteneur (A, B) puis la suite de conteneurs (B, C, D).

Dans les deux cas on utilise 3 suites continues de conteneurs mais disposés différemment. Si l'on fait le décompte du nombre de fois ou une requête doit être sectionnée pour entrer ou sortir d'une suite de conteneurs, on s'aperçoit que le groupage Gr_1 sectionne 1 fois deux requêtes (les requêtes z_2 et z_3 en D). Dans le cas du groupage Gr_2, 3 requêtes sont sectionnées (z_1, z_2 et z_3). Cette différence a un impact direct sur les équipements à placer au niveau des nœuds du réseau.

Le groupage Gr_1 paraît donc plus avantageux que le groupage Gr_2, mais Gr_1 n'est pas forcément réalisable : Gr_1 nécessite d'allouer sur les arcs BC et CD deux conteneurs simultanément ce qui n'est pas forcément possible si la capacité n'est pas suffisante. Dans le cas de Gr_2 on alloue un seul conteneur sur n'importe quel arc du réseau.

On doit donc trouver un compromis entre la capacité des câbles du réseau et le regroupement des requêtes dans des conteneurs. L'exemple présenté ne montre qu'un niveau de groupage alors que le problème général cherche à considérer un nombre arbitraire de niveaux. Pour pouvoir clarifier la notion de suite de conteneurs d'un niveau donné, nous introduisons dans la suite le modèle de tubes issu des travaux de [**HPS02**].

3.1.3. Modèle de "tubes"

Nous présentons ici une modélisation du problème qui permet de clarifier l'objectif du groupage. Celle-ci met en jeu les équipements au niveau des routeurs ainsi que les conteneurs au niveau des câbles du réseau.

Le chapitre 1 définit la notion de conteneur d'un niveau donné comme l'encapsulation d'un groupe de conteneurs d'un niveau inférieur, typiquement, l'encapsulation de plusieurs longueurs d'onde dans une bande. À partir de cette définition d'un conteneur, nous définissons la notion de "tube" comme une suite de conteneurs d'un niveau donné : on considère par exemple que la suite de bandes, traversant les nœuds intermédiaires est un tube de bandes. Un tel tube est représenté en figure 3.4 : dans cet exemple, plusieurs longueurs d'onde sont groupés (le rectangle noir) en une bande traversant le nœud central C sans être "ouverte". L'ouverture d'un conteneur signifie qu'on extrait les conteneurs de niveau inférieur pour les envoyer au multiplexeur de ce niveau inférieur. Dans notre modèle de tubes, si un conteneur est "ouvert", le tube est terminé.

Ainsi, si une bande est formée à un nœud et traverse un seul câble du réseau, on considère qu'un tube de longueur 1 a été formé. L'objectif du routage étant de minimiser le nombre de ports des brasseurs intermédiaires, pour une connexion traversant plusieurs nœuds, il s'agit, en terme de tubes, d'essayer d'utiliser les tubes les plus longs possibles. Si l'on ne se focalise que sur la longueur de ces tubes, on définirait des tubes de longueurs d'onde (chaque tube contient une seule longueur d'onde) créant ainsi autant de tubes que de requêtes unitaires, ce qui n'est pas possible car le nombre de tubes est limité par la capacité des arcs du réseau. Dans notre modèle hiérarchique, nous cherchons à créer des tubes de plus haut niveau pour agréger (ou grouper) le trafic dans des fibres ou des bandes. À l'inverse, si l'on ne définit que de longs tubes du niveau fibre, on risque de ne pas réussir à grouper assez de chemins et d'obtenir des tubes de fibres partiellement vides. Ainsi, il faut trouver un compromis entre la longueur des tubes et leur niveau hiérarchique dans le modèle.

Le problème du groupage se définit alors comme la recherche de ces tubes hiérarchiques, en cherchant à minimiser le nombre de tubes ce qui permet de réduire le brassage et le multiplexage des conteneurs d'un niveau vers un autre. En effet, le modèle du coût du réseau est

linéaire avec le nombre de tubes. Nous considérons que le coût d'un tube de conteneur d'un niveau donné est égal à la somme des coûts des OXC qui sont traversés. Dans le chapitre 6 de cette thèse, ces tubes sont implémentés dans le logiciel PORTO. On peut directement les visualiser au travers de l'interface graphique, comme le montre la figure B.7 dans l'annexe B.3.2.

3.2. Différentes approches du problème

Mentionnons tout d'abord un état de l'art qui concerne le problème général du groupage du trafic [**DR02**]. Suivant les topologies des réseaux utilisés ainsi que des requêtes sur ces réseaux, on peut distinguer les différentes méthodes d'approche du problème du groupage. Lorsque les topologies sont régulières on peut obtenir des résultats théoriques exacts. S'il s'agit de traiter des topologies quelconques, la résolution du routage et du groupage en une seule fois devient trop complexe : on préfère séparer ces deux problèmes en deux étapes qui sont résolues à la suite.

3.2.1. Topologies régulières

Même lorsque l'on suppose le routage connu, fixer le groupage reste un problème difficile à résoudre. Dans [**CRD04**], les auteurs montrent que le problème est $\mathcal{NP}$-complet dans un réseau en anneau avec l'objectif "min-max", c'est-à-dire minimisant le maximum des coûts des nœuds du réseau.

Dans [**BCM03, BC03**], on montre comment minimiser le nombre total d'OADM sur un anneau unidirectionnel où l'on groupe des flux dans des longueurs d'onde. Dans ce travail, on s'intéresse à un facteur de groupage C (le nombre de flux qui peut être groupé dans une longueur d'onde) de 4 pour du trafic *All-to-all*. Il faut noter que dans de telles hypothèses, le routage est fixé (puisqu'il est sur un anneau unidirectionnel) alors que dans d'autres formulations du groupage, on calcule le routage également. Or, dans une telle formulation, avec un coefficient de groupage C quelconque, le groupage reste un problème $\mathcal{NP}$-complet.

3.2.2. Topologies quelconques

Pour les réseaux ayant une topologie quelconque, des approches à base de programmation linéaire en nombres entiers sont données dans [**ZM02, HPS02**]. La formulation de [**ZM02**] est très détaillée, mais la complexité des calculs à réaliser est telle que les auteurs ne peuvent présenter des résultats que pour des réseaux très petits.

Dans [**UMK03**], les auteurs s'intéressent au lien qui existe entre la protection et le groupage. Dans ce cas, l'objectif est encore différent : les auteurs cherchent à minimiser les équipements au niveau des nœuds d'un réseau ayant une topologie de cycle ou de ligne avec un trafic non uniforme et non symétrique. On compte le nombre d'OADM utilisés sur ce réseau pour router les demandes, critère similaire aux critères d'évaluations de nos expérimentations de la section 3.5. Le routage est ramené à un problème de coloration des requêtes sur la ligne, puis le groupage se fait en deux phases : la première phase regroupe des chemins en les mettant bout à bout, la seconde s'appuie sur ces appariements pour calculer le groupage proprement dit. Cette approche montre que pour une topologie simple, avec un trafic considéré non régulier, le problème devient d'autant plus difficile.

Dans [**ZKW03**] le problème de conception est étudié en prenant en compte un maximum de détails liés au réseau : équipement au niveau des nœuds, capacités, possibilité d'étendre ces équipements et ces capacités. Le problème posé est celui du routage et du groupage, résolu en une seule fois, ce qui se rapproche de notre modèle qui cherche à caractériser l'influence du

routage sur le groupage, en section 3.4. Dans ce modèle, les équipements déjà présents sur le réseau sont considérés comme gratuits. Les équipements qui peuvent être ajoutés ont un coût connu. On cherche alors à calculer les extensions à ajouter au réseau, à l'aide d'une formulation complexe de programmation linéaire, les variables représentant à la fois le trafic à l'intérieur du réseau ainsi que les extensions à ajouter à celui-ci. La solution obtenue n'étant pas entière, on extrait la partie entière de la solution et on réalise un *flot de coût minimum* avec les parties des requêtes qui ne sont pas routées par cette première solution. Les auteurs proposent ensuite une heuristique de coloration pour la résolution du problème d'affectation des longueurs d'onde. Enfin, une partie expérimentale présente des résultats sur des réseaux similaires aux nôtres, la taille de ces réseaux variant de 14 à 43 nœuds.

Dans la suite, nous supposerons que les capacités de conversion des longueurs d'onde sont infinies. Les équipements du réseaux sont décrits en terme de nombre de fibres disponibles mais l'équipement des nœuds, pour sa part, est une variable du problème. Ainsi, notre objectif est de minimiser le nombre d'entrées/sorties des équipements des différents niveaux de multiplexage. Nous considérerons d'abord que le routage est préalablement fixé et nous présenterons ensuite des méthodes de routage pour le groupage[1].

3.2.3. Heuristiques de groupage

Comme nous l'avons dit précédemment, le problème du groupage est $\mathcal{NP}$-complet que ce soit à routage fixé ou dans des formulations en programmation linéaire incluant le problème du routage, proposées dans [**ZM02, HPS02, CR02**]. Ces programmes linéaires sont extrêmement difficiles à résoudre pour des réseaux de grande taille et c'est pourquoi différentes heuristiques sont proposées dans la littérature que nous résumons par la suite.

***Maximizing Single-Hop Traffic* (MST) [ZM02].** Cette heuristique très simple cherche à établir des chemins pour le trafic en avançant par "saut" et en cherchant à aller le plus loin possible, en considérant les requêtes par taille croissante. Les requêtes sont routées sur le plus court chemin entre la source et la destination, sous des contraintes de disponibilité de capacité et d'équipement des nœuds.

***Maximizing Resource Utilization* (MRU) [ZM02].** Cette heuristique cherche à maximiser le trafic moyen par câble du réseau. Le rapport entre la taille de la requête et la distance physique en nombre de sauts entre la source et la destination est utilisé pour déterminer la priorité de routage de cette requête. La procédure est en fait la même que pour *MST* et ne diffère que dans la détermination des priorités.

***Grouping lightpaths with the same destination* [LYK+02].** Cette heuristique s'applique aux réseaux utilisant 3 niveaux de conteneurs. Pour construire les conteneurs intermédiaires, les chemins ayant la même destination sont groupés ensemble de sorte que les conteneurs intermédiaires n'aient plus besoin d'être dégroupés avant leur arrivée à destination. On réalise d'abord le routage des chemins, puis on cherche à grouper les chemins de même destination dans ce niveau intermédiaire.

***Pipe filtering* [HPS02].** Si l'on prend toutes les subdivisions d'un chemin en sous-chemins (ou tubes, cf. section 3.1.3 pour le modèle en tube), on obtient un ensemble de possibilités beaucoup trop grand pour mener à la résolution d'un programme linéaire. L'objectif de cette heuristique est d'éviter les tubes trop longs qui ne seront jamais utilisés. Pour éliminer ces tubes pathologiques, on utilise une fonction d'évaluation f qui calcule, à partir de la longueur l, et du

[1] Ce travail a fait l'objet de la publication [**LPS03**].

trafic maximum t, pouvant utiliser ce tube, un grade $f(l,t)$. Tous les tubes ayant un grade plus petit qu'une certaine valeur $mingrade$ sont supprimés du programme linéaire utilisé.

3.3. Algorithme glouton de *groupage local*

Nous introduisons à présent un algorithme de *groupage local* traitant un modèle générique du groupage. À partir de ces algorithmes, nos expérimentations porteront sur le modèle spécifique à PORTO, c'est-à-dire à 3 niveaux hiérarchiques.

3.3.1. Vers un modèle récursif

À partir de la généralisation du problème du groupage de la section 3.1.2, nous proposons d'attaquer le modèle récursivement, en isolant les niveaux hiérarchiques deux par deux. En effet, il est illusoire de vouloir réaliser le groupage de tous les conteneurs en même temps : nous préférons grouper les conteneurs du niveau le plus bas, dans le niveau directement supérieur et poursuivre récursivement jusqu'à atteindre le niveau le plus haut.

Pour faciliter la description de l'algorithme de *groupage local*, nous utiliserons le modèle hiérarchique à trois niveaux, fibre, bande et longueur d'onde. Cependant, les algorithmes développés peuvent prendre en compte un nombre arbitraire de niveaux. Cependant, certains résultats issus de PORTO présenteront le cas d'un réseau à deux niveaux (fibre et longueurs d'onde).

3.3.2. Groupage local d'un nœud

Nous présentons dans cette partie un algorithme glouton pour réaliser un groupage à partir d'un routage donné. Cet algorithme a été implémenté dans le logiciel PORTO [**IMR01**].

La stratégie utilisée ici est locale : une fois le chemin fixé, on considère chaque nœud un par un. Dans un même nœud, les conteneurs d'un niveau arrivant en ce nœud sortent sur un certain câble. Si le nombre de conteneurs remplit complètement un conteneur du niveau supérieur, on peut alors les grouper ensemble.

Dans un premier temps, nous décrirons l'algorithme sur un groupage à deux niveaux : nous souhaitons grouper des longueurs d'onde dans des bandes contenant chacune 4 longueurs d'onde, pour simplifier notre exmple. La première étape de la figure 3.2 présente la situation dans un nœud du réseau. Il y a ici trois types de destinations entrant dans le nœud par les fibres A et B et sortant par les fibres C, D, et E :

- Les requêtes sortant en C (trait plein)
- Les requêtes sortant en D (pointillés courts)
- Les requêtes sortant en E (pointillés longs)

À ce stade, l'algorithme de *groupage local* se place à un nœud du réseau et cherche les regroupements possibles dans une même bande des différentes longueurs d'onde de la manière suivante :

- Il traite les fibres entrantes une à une.
- Pour chaque fibre, il sélectionne le groupe de longueurs d'onde ayant une même destination (comme [**LYK$^+$02**]).
- Il place les longueurs d'onde de ce groupe dans l'ordre, sur la fibre de sortie.
- Si 4 longueurs d'onde successives sont placées sur une bande, il constitue une bande (groupage)
- Le processus se poursuit avec le groupe de longueurs d'onde ayant une destination différente, puis change de fibre entrante.

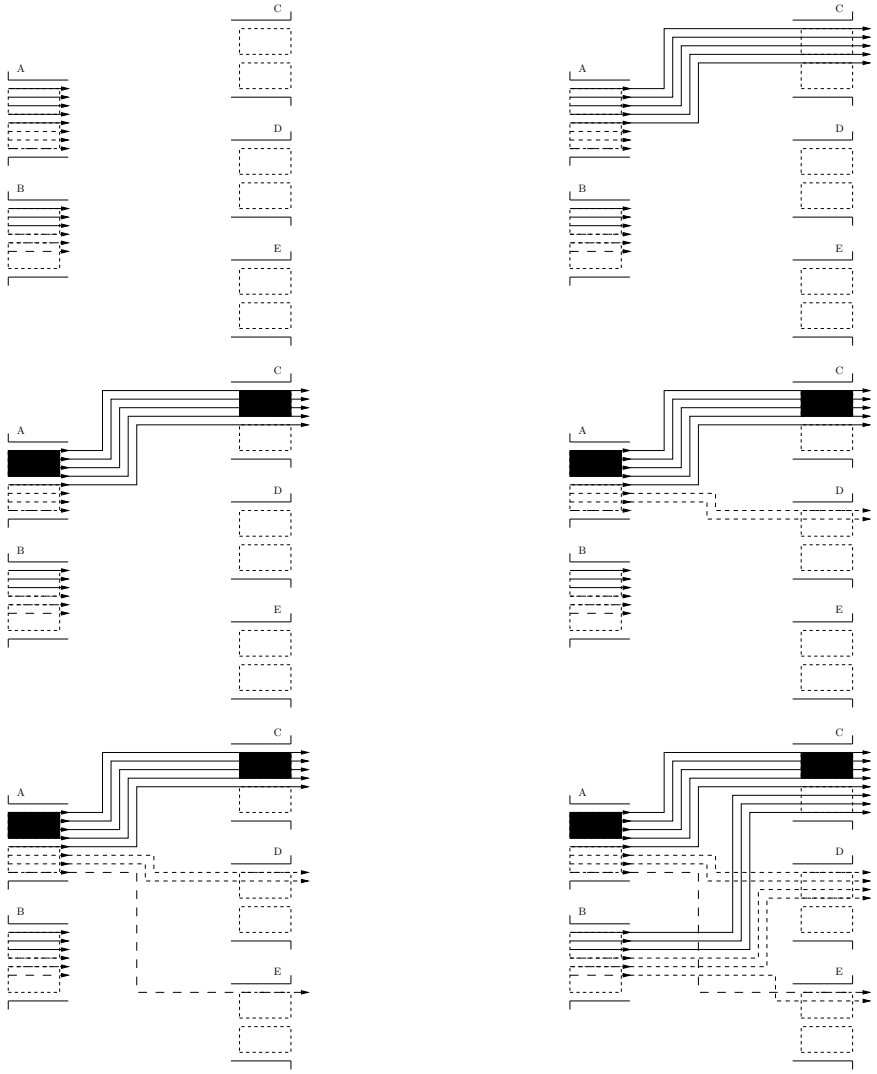


FIG. 3.2 – De gauche à droite et de haut en bas, la séquence d'allocation des longueurs d'onde sur un même nœud.

Nous présentons la décomposition de l'allocation en figure 3.2. Les rectangles en pointillés représentent les bandes qui peuvent être formées à partir de 4 longueurs d'onde. L'algorithme présenté se déroule ainsi sur cet exemple : à l'étape 2, on sélectionne les longueurs d'onde ayant pour destination C et provenant de A. Ces 5 longueurs d'onde (trait plein) sont routées sur les longueurs d'onde vides de la fibre C. À l'étape 3, on constate qu'une bande de longueurs d'onde peut être formée (rectangles noirs). À l'étape 4, on route les longueurs d'onde à destination de D et provenant de A. À l'étape 5, on route la longueur d'onde à destination de E. À l'étape 6, on route les longueurs d'onde à destination de C, D, E mais provenant de B. Ainsi, la fibre C est complétée jusqu'à sa capacité maximale (8 longueurs d'onde) et une bande a pu être formée. La fibre D est remplie à moitié, et cependant, aucune bande ne peut être formée car sur les 4 longueurs d'onde sortantes, deux proviennent de A et 2 de B.

L'état final du nœud après le déroulement de l'algorithme permet de former une bande qui passe directement de A à C. Cette bande ne passe donc pas par le multiplexeur de longueurs d'onde, dans la couche W-OXC. Pour toutes les autres longueurs d'onde, il faut passer par

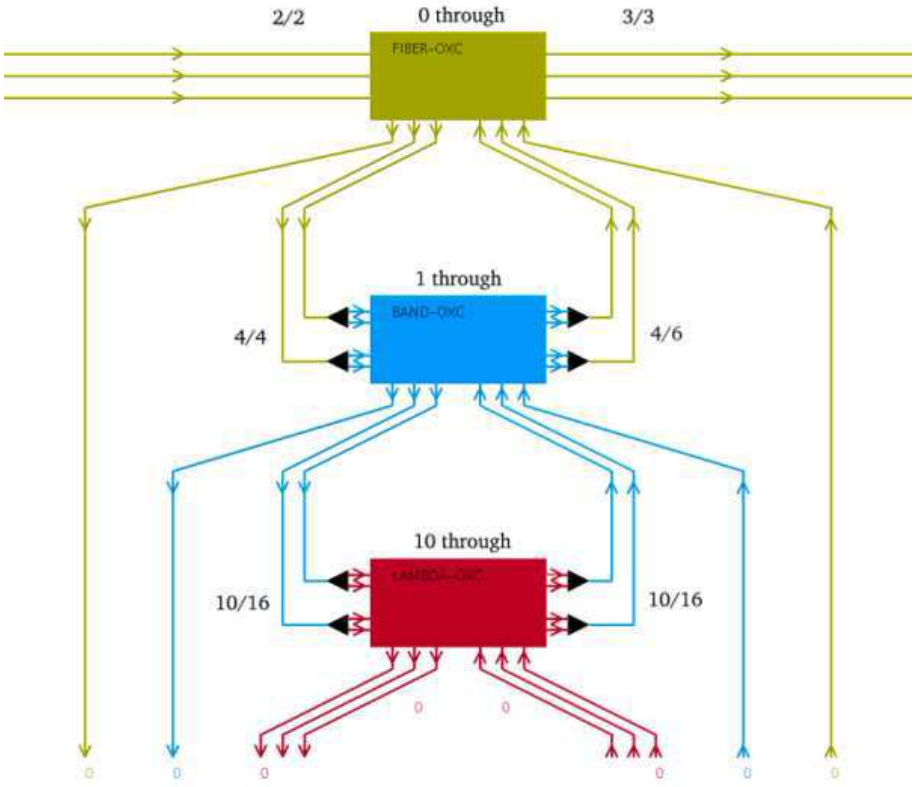


FIG. 3.3 – Configuration du brasseur pour l'exemple de la figure 3.2.

le multiplexeur de longueurs d'onde pour pouvoir former les nouvelles fibres sortantes. La figure 3.3 montre l'état du brasseur que l'on obtient après ce groupage. Pour chaque couche, la notation x/y indique que x conteneurs provenant de l'ouverture du conteneur supérieur arrive à ce niveau, sur un total possible de y. Ainsi, au niveau B-OXC, 4 bandes doivent être traitées par le B-OXC. Sur ces 4 bandes, 3 descendent au niveau W-OXC et 1 bande reste au niveau B-OXC (la bande groupée). Les longueurs d'onde restantes sont brassées au niveau W-OXC, et remontent pour être encapsulées dans des bandes. Enfin, 4 bandes sur 6 possibles sont formées et sont à leur tour multiplexées dans les 3 fibres sortantes.

3.3.3. Groupage résultant du groupage local

L'algorithme glouton du *groupage local* réalise le groupage sur chaque nœud et localement, c'est-à-dire sans tenir compte des nœuds voisins. Une fois le groupage interne des nœuds réalisé, on peut déduire le groupage du réseau. Plus précisément, si plusieurs groupages locaux se succèdent, comme montré en figure 3.4, le groupage est meilleur puisqu'il assure la continuité d'un conteneur (ici une bande) sur plusieurs sauts consécutifs. On réduit donc d'autant plus le nombre d'entrées/sorties nécessaires au niveau W-OXC.

L'efficacité de l'algorithme reste tout de même limité. Le nombre de connexions entrantes dans les nœuds du réseau perturbe la régularité des groupage locaux. Alors qu'un conteneur est formé à un nœud, il peut être cassé au nœud suivant si à ce nœud une longueur d'onde doit par exemple s'arrêter. Cependant, l'algorithme fonctionne relativement bien lorsque les requêtes

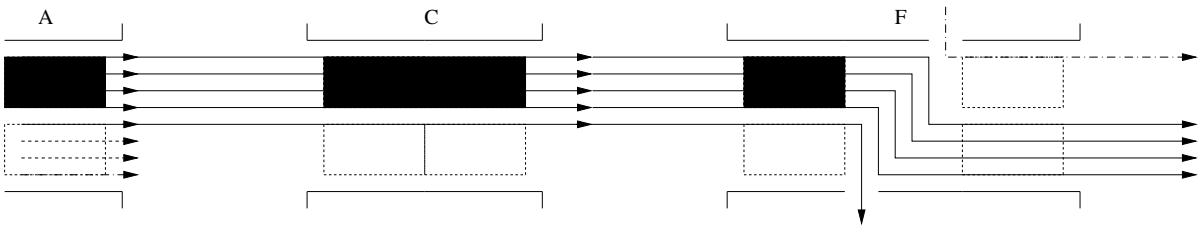


FIG. 3.4 – Exemple de séquence de groupages locaux.

sont de grande taille : l'accroissement du nombre de longueurs d'onde par requête permet à l'algorithme d'allouer des séries consécutives de longueurs d'onde sur les nœuds. Ces séries étant étendues, il est alors assez facile de regrouper les longueurs d'onde par bandes.

3.4. Le routage pour le groupage

L'algorithme de groupage local utilise un routage déjà calculé au préalable. Dans cette section, nous proposons d'adapter le routage des requêtes pour faciliter le processus de groupage. Les résultats expérimentaux permettront de mesurer l'influence du routage sur le groupage.

3.4.1. L'influence du routage sur le groupage

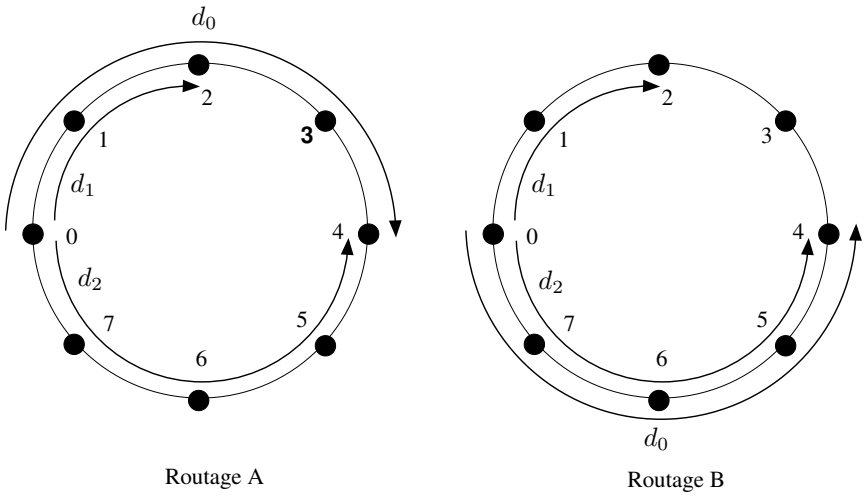


FIG. 3.5 – 2 stratégies de routage.

Nous montrons en figure 3.5 quelle peut être l'influence du routage sur le groupage. On suppose, dans cet exemple, que ce réseau en forme d'anneau est bidirectionnel et possède une capacité d'une fibre dans chaque direction. Chaque fibre peut contenir deux bandes.

Trois requêtes de la taille d'une bande, d_0, d_1 et d_2 sont routées avec deux stratégies de routage différentes (A) et (B). Le routage (A) nécessite l'utilisation d'un B-OXC au nœud 2 alors que le routage (B) nécessite qu'un F-OXC au nœud 2 car aucune bande n'a besoin d'être extraite de la fibre provenant du nœud 1. Dans cet exemple les requêtes d_0 et d_2 ont la même source et la même destination, ce qui reste rare dans nos instances (on considère alors qu'il

s'agit de la même requête). En pratique, un calcul du multiflot, suivant la technique utilisée, peut conduire à router une seule requête sur plusieurs chemins, ce qui nous ramène au cas présenté.

Compte tenu de l'influence du routage sur le groupage, nous proposons plusieurs heuristiques de routage qui permettent de réaliser un groupage efficace, puis nous les validons expérimentalement sur nos réseaux optiques présentés en annexe A.2. Ces heuristiques utilisent les algorithmes de calcul du multiflot présentés au chapitre 2. L'expression du multiflot utilisée est présentée en section 2.1.2 (équations 2.9 à 2.13). Il faut cependant noter que ces heuristiques ne réalisent pas le groupage proprement dit : elles sont destinées à faciliter un algorithme de groupage qui aura lieu après le routage. L'avantage principal d'un tel découpage (et qui est réalisé dans le PORTO, présenté au chapitre 6) est de pouvoir développer différents modules de routage et de groupage interchangeables et pouvant être librement combinés.

3.4.2. Routage *Biggest first* et *Filling threshold*

L'heuristique *Biggest first* est un algorithme glouton qui route les requêtes prises par ordre décroissant de taille. Suivant cet ordre, on calcule un flot pour chaque requête et on réserve l'espace nécessaire sur le réseau pour son acheminement, espace déduit de l'espace restant pour le prochain calcul. L'idée principale de cet algorithme est d'éviter un éclatement des requêtes principales qui les rendraient difficile à grouper par la suite.

La deuxième heuristique présentée ici, *Filling threshold*, cherche à éviter le possible éclatement du routage d'une autre manière : à chaque étape de *Filling threshold*, les fibres dont la capacité utilisée dépasse un certain seuil sont volontairement saturées à la taille d'une fibre. Si l'on définit T comme le seuil, avec $0 < T < W$, on complète une fibre dès que :

$$\exists k \in \mathbb{N} / \; kW + T \leq \sum_{z \in Z} x_{e,z} \leq (k+1)W, \; \forall e \in E$$

La saturation se fait par l'ajout de longueurs d'onde fictives, associées à la requête considérée jusqu'à l'arc e soit occupé par $(k+1)W$ longueurs d'onde. Ces longueurs d'onde fictives ont toutes les chances d'être groupées dans une fibre complète lors de l'algorithme de groupage, le *groupage local* dans nos expérimentations. Par ailleurs, les prochains routages devront utiliser un réseau dont les capacités ont été modifiées et où les arcs ayant subi une saturation possèdent un espace résiduel multiple de W, ce qui leur permettra aussi d'avoir une meilleure chance d'être groupés.

3.4.3. Routage *Filter*

Le routage filtré s'inspire de la méthode *Filling threshold* et tente de reproduire l'algorithme glouton à tous les niveaux de granularité des conteneurs du réseau. L'algorithme itère sur chaque niveau de granularité, en l'occurence, dans le cas des réseaux WDM, sur les niveaux fibre, bande et longueur d'onde. L'algorithme tente alors pour chaque niveau de router des demandes qui pourront compléter un niveau totalement. Ainsi, à chaque itération concernant un niveau l, on réalise les opérations suivantes :

- On sélectionne les requêtes ayant une taille minorée par $T_{request}(l)$, les seules susceptibles de remplir un conteneur de taille l.
- On route ces requêtes sur le réseau.
- On complète les chemins de conteneurs remplies à $(T_{path}(l)/W_l)\%$ ou plus.
- On met à jour l'espace restant sur le réseau (variable $c(e)^{paths}$).

Algorithme 5 Heuristique du routage *Filter*.

Entrée: $G = (V, E)$, $|V| = n$, $|E| = m$, Z
Sortie: Un ensemble de chemins $P(z)$ pour Z

1: {Initialisation} $\forall e \in E$, $c(e)^{paths} = 0$, niveau $l = 0$ {Le niveau fibre}
2: **tant que** $\exists z \in Z$ t.q. $size(z) \neq 0$ **faire**
3: **répéter**
4: creation = false, $Z_l = \{z_i \in Z \text{ / } size(z_i) \geq T_{request}(l)\} = \{z_1, \ldots, z_k\}$
5: Construire le programme linéaire (2.9 - 2.13) avec $Z = Z_l$ et $\forall e \in E$, $c(e) = c(e)^0 - c(e)^{paths}$
6: Résoudre le programme linéaire et sauver les chemins associés à Z_l dans $P_l = \{p_1, \ldots, p_k\}$
7: **pour tout** $p_i \in P_l$ t.q. $size(p_i) \geq T_{path}(l)$ **faire**
8: Séparer p_i en sous chemins $\{p_i^1, \ldots, p_i^m\}$ t.q. $\forall j \in \{1, \ldots, m\}, size(p_i^j) \leq W_l$ et m minimum
9: **pour tout** $p_i^j \in \{p_i^1, \ldots, p_i^m\}$ **faire**
10: **si** $\forall e \in p_i^j$, $c(e)^0 - c(e)^{paths} > size(p_i^j)$ **alors**
11: Compléter, si possible, avec des chemins fictifs p_i^j pour atteindre W_l et ajouter p_i^j dans $P(z)$
12: $\forall e \in p_i^j$, $c(e)^{paths} = c(e)^{paths} + size(p_i^j)$
13: $size(z_i) = size(z_i) - size(p_i^j)$
14: creation = true
15: **fin si**
16: **fin pour**
17: **fin pour**
18: **jusqu'à non** creation
19: $l = l + 1$
20: **fin tant que**

L'heuristique détaillée est présentée dans l'algorithme 5. Les notations suivantes sont utilisées : l est le niveau de conteneur considéré : fibre, bande, longueurs d'onde. $T_{request}(l)$ et $T_{path}(l)$ sont les paramètres de seuil qui contrôlent les requêtes et les chemins sélectionnés, pour chaque niveau l. $c(e)^{paths}$ est l'espace occupé par les chemins créés au fur et à mesure de l'algorithme, compté en longueurs d'onde ; $c(e)^0$ est l'espace initial des câbles du réseau et $c(e)$ est l'espace disponible courant.

Les opérations détaillées précédemment sont effectuées pour un niveau donné l et ce niveau ne change pas tant que des chemins sont créés et sauvés par l'algorithme en ligne 11. En effet, à cause de l'ajout des chemins fictifs, il se peut que le traitement des chemins p_i^j dans la boucle de la ligne 9 ne soit pas possible (on ne peut plus les router tel que le multiflot l'a calculé car on a "gaspillé" de l'espace pour des chemins précédents). Dans ce cas, il faut recalculer un multiflot complet pour traiter ces chemins, en conservant la granularité courante, l. Lorsqu'à cette granularité, aucun chemin n'est sauvé dans $P(z)$, $\forall z \in Z$, on passe à la granularité suivante (ligne 19).

Grille	Requêtes	c(e)	Multi	Single	*Filling threshold*	*Filter*
5×3	15⇒7	4	1	1.8	2.0	1.46
5×4	20⇒10	5	2.6	6.1	5.7	4.6
5×5	25⇒12	6	8.3	×	13.3	9.1
5×6	30⇒15	7	54	×	27.5	88.7
5×7	35⇒18	9	140.6	×	59.5	163.3

TAB. 3.1 – Temps d'exécution (s), $size(z) = 10, \forall z \in Z$ (× = Timeout).

3.5. Résultats expérimentaux

Nos expérimentations sont basées sur deux ensembles de réseaux. Le premier ensemble comporte des grilles dont le trafic est généré aléatoirement et permet de tester les performances de nos algorithmes (section 3.5.1). Le deuxième ensemble est basé sur certains réseaux réels présentés en annexe A.2. Ce deuxième ensemble permettra de comparer la qualité de nos différents algorithmes de routage pour le groupage.

Tous les tests effectués ont nécessité de fixer des paramètres de nos algorithmes à des valeurs arbitraires (qui peuvent ensuite être adaptées aux caractéristiques d'un réseau ciblé). À partir du nombre de longueurs d'onde par fibre, spécifié pour chaque réseau en section 1.1, les paramètres de seuil ont été fixés pour les différents algorithmes. Par exemple, lorsque le réseau possède 32 longueurs par fibre on fixe les différents paramètres ainsi : pour *Filling threshold*, $T = 28$; pour *Filter*, $T_{request} = 28, 6, 1$ et $T_{path} = 26, 6, 1$. La différence entre $T_{request}$ et T_{path} permet de favoriser l'ajout des chemins fictifs : on n'autorise que les requêtes conséquentes (supérieures à 28, pour le niveau fibre) mais on s'autorise à grouper des chemins moins garnis (supérieurs à 26, pour le niveau fibre). Évidemment, un usage abusif du paramètre T_{path} peut conduire à l'échec d'une solution à cause d'un gaspillage excessif des ressources.

Nous utilisons l'algorithme glouton de groupage présenté en section 3.3.2 pour réaliser le groupage post-routage. L'évaluation du résultat est principalement fait en comptabilisant le nombre de ports au niveau W-OXC nécessaire pour le réseau dans son ensemble. Cette valeur, notée UW pour *Ungroomed Wavelengths*, comprend le nombre de longueurs d'onde qui traversent la couche W-OXC ainsi que les longueurs d'onde qui doivent sortir du réseau à un nœud (*dropped wavelengths*).

Les expérimentations effectuées dans la suite comparent les algorithmes présentés à un multiflot classique, noté *Multi*, et à un monoflot, noté *Single*, que nous avons choisi de présenter en section 4.2.2 dans le cadre de la protection. L'algorithme noté *Biggest* correspond à *Biggest first*, et *Filling* correspond à *Filling threshold*. L'algorithme noté *Order* dans les différentes figures est une heuristique qui route les requêtes dans l'ordre décroissant de leur taille.

3.5.1. Performances

La table 3.1 présente le temps d'exécution moyen pour différents réseaux ayant une topologie de grille. La demande est artificiellement construite selon une répartition *All-to-Half* ce qui signifie que chaque nœud envoie une demande de taille fixée (ici 10 longueurs d'onde) vers chaque nœud d'un sous-ensemble comportant la moitié des nœuds du réseau. Etant donné que le nombre de nœuds augmente pour les différentes grilles, la capacité des arcs doit aussi être augmentée pour ne pas surcontraindre le problème. Nous avons fixé arbitrairement la taille des

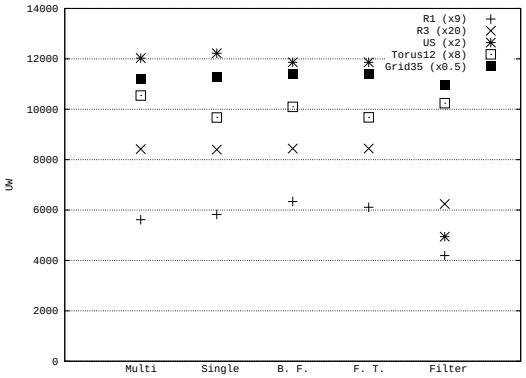


FIG. 3.6 – UW avec 100% des requêtes.

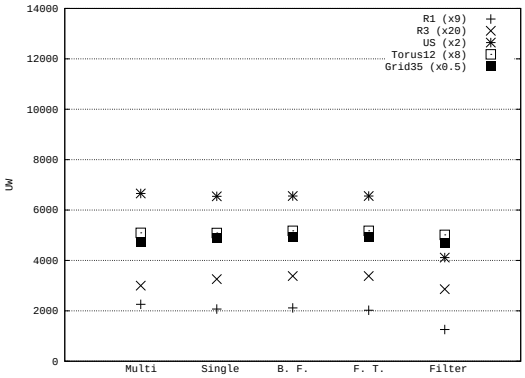


FIG. 3.7 – UW avec 50% des requêtes.

câbles (exprimée en nombre de fibres) au quart du nombre de nœuds, i.e. $e_c = \lceil \frac{n}{4} \rceil$. Les différentes expérimentations montrent que ces algorithmes sont rapides, même avec un nombre de requêtes élevé : 630 requêtes sont routées sur 35 nœuds en moins de 3 minutes sur un PENTIUM™ IV.

3.5.2. Résultats de groupage

L'algorithme *Filter* permet d'économiser un nombre important de ports sur la couche WOXC, et particulièrement pour les réseaux à topologie réelle, comme pour EU ou R_3. La figure 3.6 montre que l'algorithme arrive à grouper jusqu'à 50% des longueurs d'onde non groupés par les autres algorithmes. Les algorithmes *Filling threshold* et *Biggest first* permettent d'économiser des longueurs d'onde mais dans des proportions qui restent minimes. Notons la contre performance obtenue sur $GRID_{12}$ et un résultat médiocre sur $GRID_{35}$ dus au fait que ces deux réseaux ont un trafic du type *All-to-All* c'est-à-dire que chaque nœud du réseau communique avec tous les autres nœuds du réseau, ce qui représente la matrice de trafic la plus difficile à gérer.

3.5.3. Robustesse à la charge

Les figures 3.8 et 3.9 confirment que l'algorithme *Filter* reste efficace même lorsque la charge du trafic augmente. Pour toutes les valeurs de la charge $load$, le facteur multiplicatif de la taille de l'instance de communication, plus de 25% d'UW est économisé. Notons que dans le cas de R_1, en figure 3.8, l'algorithme *Filter* ne peut plus trouver de solution pour une charge supérieure à 0.72. La capacité sur le réseau vient alors à manquer, du fait du gaspillage des longueurs d'onde qui doit être fait pour améliorer la phase de groupage.

Nos expérimentations concernent aussi des grilles 4x4 supportant un trafic généré aléatoirement. Les requêtes de communication sont tirées aléatoirement, à la fois pour le choix du nœud source et du nœud de destination, mais aussi pour leur taille (tirage uniforme). Pour chaque charge $load$, les expérimentations ont été effectuées 150 fois, ceci afin de gommer les variations dues au caractère aléatoire de chaque instance et de présenter la moyenne des résultats (méthode dite d'évaluation par "réplication"). La figure 3.10 présente l'évolution de UW lorsque la charge augmente, en terme de taille des requêtes. L'utilisation de l'algorithme *Filter* permet d'absorber cette augmentation donnant une courbe quasi constante ce qui montre que

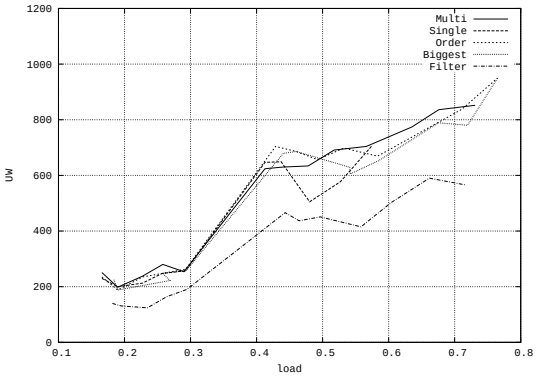


FIG. 3.8 – UW pour R_1 en fonction de l'augmentation de la charge.

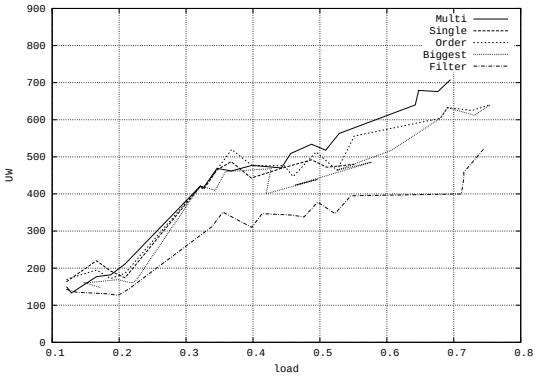


FIG. 3.9 – UW pour R_3 en fonction de l'augmentation de la charge.

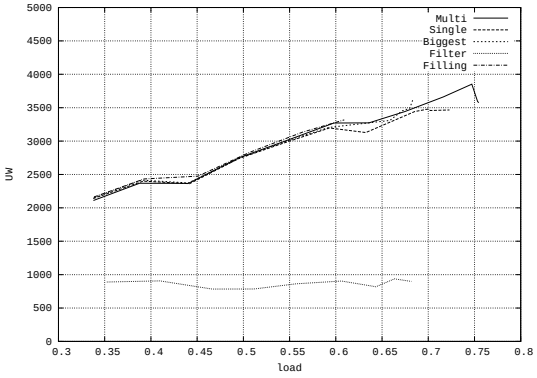


FIG. 3.10 – UW pour une grille 4x4 en fonction de l'augmentation de la taille des requêtes.

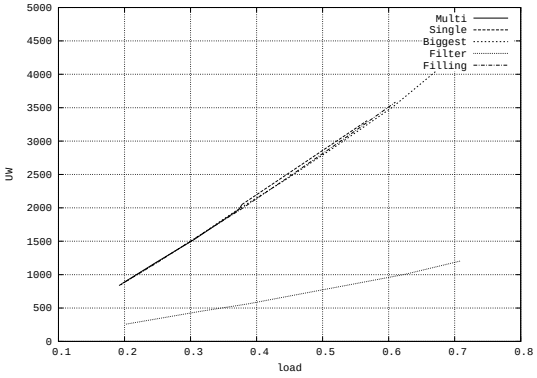


FIG. 3.11 – UW pour une grille 4x4 en fonction de l'augmentation du nombre de requêtes.

l'algorithme arrive à grouper correctement un trafic augmentant en tirant partie de plus en plus des ressources gaspillées. Dans l'expérimentation de la figure 3.11, la dépendance devient linéaire au nombre de requêtes à considérer. *Filter* arrive à utiliser six fois moins de longueurs d'onde au niveau W-OXC que les autres algorithmes et atteint le point de saturation le plus haut, 0.7 : l'algorithme se comporte correctement avec un grand nombre de requêtes de tailles très différentes.

Conclusion

Dans ce chapitre nous avons présenté le problème du groupage dans les réseaux optiques. À partir de la définition du problème dans les réseaux optiques, nous avons décrit un modèle générique de tubes hiérarchiques permettant de généraliser la formulation du problème. Ce modèle permet alors de définir des algorithmes pouvant travailler sur un nombre quelconque de niveaux hiérarchiques.

Nous avons proposé un algorithme glouton de groupage travaillant sur l'encapsulation de plusieurs niveaux. À partir de cette heuristique, nous avons étudié l'influence du routage sur le groupage et nous avons proposé plusieurs heuristiques favorisant le processus de groupage à partir du calcul de routages. Nous expérimentons les différentes heuristiques et montrons que l'algorithme de routage *Filter* permet d'obtenir des résultats significatifs, à la fois sur des réseaux ayant une topologie de grille et sur des réseaux réels. Le trafic est mieux groupé ce qui se traduit par un nombre de longueurs d'onde non groupés plus bas (jusqu'à 50% de gain). Enfin, l'algorithme *filter* réagit correctement à l'augmentation de la taille ou du nombre des requêtes.

Nous concluons ce chapitre en rappelant qu'il est primordial d'avoir plus de capacité disponible que nécessaire à un simple routage si l'on veut réussir à grouper le trafic. Il faut avoir de la place pour ne pas remplir complètement tous les conteneurs et éviter d'avoir à les ouvrir à cause d'une seule longueur d'onde qui a besoin de sortir d'un nœud intermédiaire. Cela permet aussi d'utiliser des routes adaptées et pas forcément les plus courtes. Il reste hélas beaucoup de travail à faire pour quantifier combien de place est nécessaire pour un type de trafic donné et pour améliorer les résultats théoriques (même pour du trafic *All-to-All* sur un réseau maillé).

Suite à la description de la problématique du groupage, nous présentons dans le chapitre 4 nos travaux concernant la protection dans les réseaux optiques.

CHAPITRE 4

Algorithmes de protection

Le trafic sur un réseau de cœur peut atteindre des débits de plusieurs terabits par secondes. La panne d'un lien du réseau, par exemple une coupure physique d'une fibre lors de travaux de voirie ou la panne d'un équipement optique n'est pas acceptable pour un opérateur qui souhaite garantir à ses clients une certaine qualité de service. Le volume des débits concernés et le souci de garantie de la connectivité rendent encore plus critiques les problématiques de réseau tolérant aux pannes, c'est-à-dire possédant des mécanismes permettant de restaurer un flux subissant une panne.

La tolérance aux pannes est un problème lié aux chapitres précédents de cette thèse, et notamment le routage (chapitre 2) dans les réseaux WDM. En effet, la prise en compte des pannes contraint les routes qui acheminent les requêtes. Cela demande l'utilisation de plus de ressources en terme de capacité. De plus, les modules de groupage doivent alors utiliser ce routage imposé et ne sont plus libres de réaffecter les routes pour optimiser le groupage.

Nous commençons par définir le modèle de protection auquel nous nous intéressons et à nous positionner par rapport aux autres techniques existantes utilisant des modèles ou des objectifs différents. Nous présentons ensuite des méthodes simples pour un premier calcul de protection, puis nous introduisons un modèle plus complexe utilisant la programmation linéaire. La description de ce modèle montre comment économiser des ressources réseaux en partageant des longueurs d'onde pour les chemins de protection. Nous pointons alors les limites de ce modèle et nous proposons une adaptation de ce dernier, ce qui nous permet d'obtenir des résultats plus performants en terme d'allocation de ressources. Cette adaptation permet d'utiliser les méthodes de résolution de programmes linéaires fractionnaires sur la relaxation du problème. Ensuite, nous déduisons la solution du problème entier à l'aide d'une technique d'arrondis aléatoires. Enfin, nous évaluons la qualité de notre démarche par des expérimentations et des comparaisons avec d'autres méthodes connues.

4.1. Algorithmes connus pour la restauration et la protection

En section 1.3.4 nous avons présenté la problématique de la protection. À partir de cette classification, nous présentons brièvement quelques approches du problème et les particularités des modèles utilisés permettant d'obtenir différentes stratégies de protection [**Wu95**].

On distingue en général l'approche proactive de l'approche réactive lorsque l'on parle de protection. L'approche réactive met en œuvre des algorithmes qui doivent réagir à une panne et altérer les routes du réseau pour traiter cette panne [**YJ02**]. Il n'est donc pas garanti que lors d'une panne, une connexion puisse être rétablie. À l'inverse, l'approche proactive réserve de la capacité (ou même des routes) pour une requête donnée, pour tout cas de panne. En fait,

la différenciation de ces deux politiques intervient lorsqu'on élabore des algorithmes *online*, c'est-à-dire qui décident d'une politique en temps réel lors de l'attribution d'une route à une nouvelle requête sur le réseau. Dans l'approche proactive, une requête peut être refusée s'il est impossible de garantir la disponibilité d'un chemin de secours. Dans ce cas, les objectifs considérés sont généralement de maximiser le nombre de requêtes acceptées sur le réseau, comme dans [**FS03**] où l'on utilise des techniques de simulation pour tester différentes politiques. On évalue alors, outre la probabilité de blocage, l'utilisation effective du réseau puisque l'on réserve des ressources pour protéger les chemins principaux mais aussi la probabilité de réaffectation i.e la mesure de l'impact sur tout le réseau lorsqu'il y a une restauration de lien. En fait, les algorithmes de protection pour une arrivée du trafic connue dynamiquement sont en général simples car ils possèdent des contraintes temps réel fortes [**MM01**]. Dans nos hypothèses de protection nous nous plaçons toujours dans le cadre d'algorithmes *offline* qui connaissent donc à l'avance l'ensemble des requêtes à satisfaire.

De nombreux travaux utilisent des techniques découpant naturellement la protection en deux phases : une phase d'allocation des chemins principaux, suivie d'une phase de protection, avec éventuellement un rebouclage sur la première phase pour améliorer la qualité globale de la solution. Ainsi, dans [**ACB97**], on calcule simplement le routage principal avec un algorithme de plus courts chemins, puis on effectue un reroutage par des chemins disjoints en utilisant une métaheuristique du type *tabou*. La difficulté d'une telle technique réside dans le fait qu'il faut obtenir des chemins disjoints et que ce problème est $\mathcal{NP}$-complet à partir de deux chemins disjoints dans un graphe orienté [**FHW80, Jar02**]. Dans [**AN01**] on propose une méthode utilisant un système de pénalité : chaque arête est valuée suivant son appartenance ou pas au chemin principal. Ainsi, en incluant cette pénalité dans la fonction objectif, on pousse les chemins de secours à être disjoints des chemins principaux. Cependant, dans cette hypothèse, on perd la garantie d'une protection du type $1:1$ à 100%.

Dans [**FS03, BW00**], on s'intéresse à des algorithmes de restauration qui calculent un routage pour chaque panne possible sur le réseau (reroutage global). Comme nous l'avons déjà mentionné, ces techniques permettent de minimiser l'espace nécessaire au routage mais ne sont pas forcément réalistes dans le sens où un nombre important de brasseurs peuvent avoir à être reconfigurés. En fait, dans la majorité des cas, on s'intéresse à des techniques de protection comme dans [**DSS03, CB98, BW00**] pour la protection $1:1$. Enfin, d'autres approches directement issues de la théorie des graphes, par exemple dans [**EHS00**] qui étudie la couverture par cycle des arcs du réseau, ou inversement dans [**HO01, BCCT01**] où l'on couvre optimalement un réseau ayant une topologie de cycle pour un trafic *All-to-All*.

Lorsque l'on s'intéresse à des techniques de protection $M:N$, il est assez fréquent de modéliser le problème sous la forme d'un programme linéaire entier surtout pour modéliser les exclusions entre les chemins principaux et de protection, qui doivent être disjoints [**SSS02**]. Dans [**IMG98**] on utilise cette formulation sous la forme d'un multiflot classique et permettant de partager la capacité de protection. La formulation demeure cependant compliquée et difficile à mettre en œuvre dans des expérimentations pratiques. S'il n'y a pas de contrainte de capacité, la formulation linéaire tente de résoudre un problème de conception prenant en compte la protection et qui calcule les capacités du réseau. Dans [**HK99**], on propose un algorithme glouton d'affectation des chemins, utilisant une fois encore une variante des plus courts chemins. On route chaque longueur d'onde de chaque demande par un calcul de plus courts chemins, puis, pour garantir l'exclusion des chemins de longueurs d'onde avec les chemins de protection, on pose des longueurs infinies sur les arêtes utilisées pour le chemin principal. Un autre algorithme est proposé dans la deuxième partie de [**HK99**] qui utilise le point de vue inverse : au fur et à

Algorithme 6 Reroutage global.

Entrée: $G = (V, E)$, Z, $c(e)$, $\forall e \in E$
Sortie: Un ensemble de routes principales $\mathcal{R}(z)$, un ensemble de routes $\mathcal{R}(z, e)$ pour chaque cas de panne $e \in E, \forall z \in Z$

1: Calculer un multiflot avec G, Z et $c : e \rightarrow \mathbb{N}$
2: Sauver les chemins dans $\mathcal{R}(z)$
3: **pour tout** $e \in E$ **faire**
4: $G_e = (V, E \backslash \{e\})$
5: Calculer un multiflot avec G_e, Z et $c : e \rightarrow \mathbb{N}$
6: Sauver les chemins dans $\mathcal{R}(z, e)$
7: **fin pour**

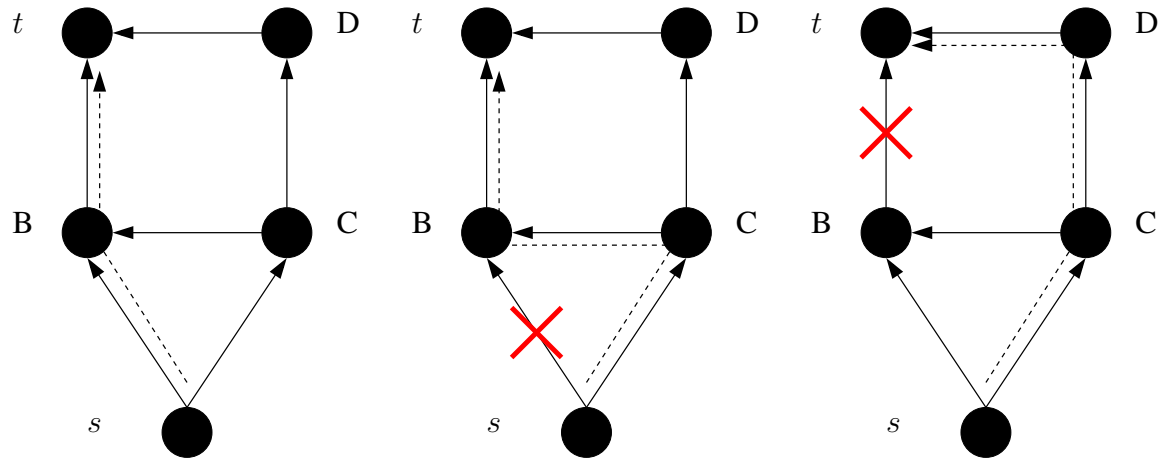


FIG. 4.1 – Exemple de reroutage global pour la requête (s, t)

mesure qu'on établit le routage principal et le routage de protection, les longueurs d'onde de protection sont considérées comme "gratuites" pour un autre chemin de protection éventuel. Ainsi, l'algorithme du plus court chemin peut utiliser cette arête ayant une longueur nulle, ce qui favorise la réutilisabilité des arêtes de protection et donc le partage entre les chemins de protection.

4.2. Présentations détaillés d'algorithmes de protection

4.2.1. Reroutage global

Le reroutage global permet de calculer une borne supérieure de dimensionnement d'un réseau. L'algorithme est très simple : il met en œuvre un de calcul de multiflot qui est répété pour tous les cas de pannes (algorithme 6). Le principal inconvénient d'une telle technique est que le calcul de multiflot étant un problème $\mathcal{NP}$-complet, le répéter $|E|$ fois pour obtenir la solution totale est rédhibitoire. Par ailleurs, de nombreuses tables de routage doivent être conservées en tous les nœuds du réseau et toute panne sur le réseau impacte tous les nœuds qui doivent utiliser une nouvelle table de routage.

Lorsque l'on a calculé tous les routages pour tous les cas de pannes possibles, l'allocation des ressources du réseau peut être déduit en réservant sur chaque arc e la valeur maximum du

flot passant par e pour tous les routages calculés. Ainsi, on garantit que les ressources réservées sont suffisantes pour écouler tous ces routages. La figure 4.1 présente le routage de la requête unitaire (s, t) qui passe par B (schéma de gauche). Pour les deux cas de pannes possibles, nous présentons deux solutions optimales : (s, C, B, t) et (s, C, D, t). Ces deux routages sont d'une longueur de 3 et sont donc optimaux pour chaque sous-problème correspondant à une panne. Cependant, lorsqu'on calcule le nombre de ressources unitaires allouées en prenant le maximum des valeurs pour toutes les pannes possibles, on obtient, dans cet exemple, une allocation de 7 (1 unité par arc du réseau). Cependant, si pour les deux cas de pannes on utilise le chemin (s, C, D, t), on réduit alors l'allocation à 5. L'exemple montre pourquoi la solution globale n'est pas optimale puisque les optimisations de chaque cas de panne sont indépendantes : on n'obtient donc qu'une borne supérieure du problème.

4.2.2. Modélisation de la protection $1 : 1$ en programmation linéaire

Lorsqu'on cherche à mettre en œuvre les modèles de programmation linéaire pour la protection, on peut utiliser des modèles simples issus des programmes linéaires représentant des multiflots présentés en section 2.1.2. La protection $1 : 1$ utilisant deux chemins disjoints, il est assez naturel de chercher à calculer un monoroutage pour le chemin principal et un deuxième monoroutage pour le chemin secondaire, en s'assurant qu'ils sont disjoints. La contrainte d'exclusion entre les deux monoflots est assez facile à mettre en œuvre. Lorsque l'on s'attaque ensuite à l'expression de la protection $M : N$, la contrainte d'exclusion ne s'exprime plus aussi simplement.

Nous présentons tout d'abord les versions adaptées des programmes linéaires élaborés en section 2.1.2 pour le calcul d'un monoroutage. Le programme linéaire 5 donne l'expression de ce monoroutage sous sa forme *sommet-arc*, alors que le programme linéaire 6 le donne sous sa forme *arc-chemin*. Nous rappelons brièvement les notations utilisées en section 2.1.2 : $x_{e,z}$ est la variable de flot pour l'arc e et la requête z. On note $add(v, z)$ et $drop(v, z)$ le flot entrant et sortant au nœud v pour la requête z. W_f est la multiplicité d'une fibre, $\delta^+(v)$ et $\delta^-(v)$ sont les ensembles d'arcs sortant et entrant pour le nœud v et $nb_f(e)$ est la variable contrôlant le nombre de fibres allouées pour l'arc e et $c(e)$ sa capacité en nombre de longueurs d'onde.

Des notations supplémentaires sont introduites ici : la constante C est utilisée pour forcer les variables $y_{e,z}$ à 1 lorsque $x_{e,z}$ est non nul. En pratique, on peut fixer C à une valeur supérieure à la taille de la demande maximum, i.e. $max_{z \in Z} size(z)$.

Programme linéaire 5 (monoroutage *sommet-arc*)**.**

$$\forall e \in E,\ \sum_{z \in Z} x_{e,z} \leq W_f \times nb_f(e) \tag{4.1}$$

$$\forall e \in E,\ W_f \times nb_f(e) \leq c(e) \tag{4.2}$$

$$\forall v \in V, \forall z \in Z,\ \sum_{e^- \in \delta^-(v),\ e^+ \in \delta^+(v)} x_{e^-,z} - x_{e^+,z} = drop(v, z) - add(v, z) \tag{4.3}$$

$$\forall v \in V, \forall z \in Z,\ \forall e \in E,\ x_{e,z} \leq C \times y_{e,z} \tag{4.4}$$

$$\forall z = (s, t) \in Z,\ \forall v \in V \backslash \{s, t\},\ \sum_{e^+ \in \delta^+(v)} y_{e^+,z} \leq 1 \tag{4.5}$$

$$\forall e \in E, \forall z \in Z,\ x_{e,z},\ y_{e,z},\ nb_f(e) \quad \textit{entiers} \tag{4.6}$$

$$\textit{Min} \sum_{e \in E} nb_f(e) \tag{4.7}$$

Programme linéaire 6 (monoroutage *arc-chemin*)**.**

$$\forall z \in Z,\ \sum_{p \in \mathcal{P}_z} \delta^A_{p,z} = 1 \tag{4.8}$$

$$\forall e \in E,\ \sum_{z \in Z,\ p \in \mathcal{P}_z,\ e \in p} size(z) \times \delta^A_{p,z} \leq W_f \times nb_f(e) \tag{4.9}$$

$$\forall e \in E,\ W_f \times nb_f(e) \leq c(e) \tag{4.10}$$

$$\forall z \in Z,\ \forall p \in \mathcal{P}_z,\ \delta^A_{p,z},\ nb_f(e) \quad \textit{entiers} \tag{4.11}$$

$$\textit{Min} \sum_{e \in E} nb_f(e) \tag{4.12}$$

La taille des programmes linéaires, par rapport à celle du chapitre 2, est alors modifiée dans la proportion suivante : le programme linéaire 5 utilise $\mathcal{O}(|E||Z| + |E|)$ variables et $\mathcal{O}(|V||E||Z|+|V||Z|+|E||Z|+|E|)$ contraintes et le programme linéaire 6 comporte $\mathcal{O}(|\mathcal{P}_Z||Z|+|E|)$ variables et $\mathcal{O}(|\mathcal{P}_Z||Z| + |Z| + |E|)$ contraintes, avec $|\mathcal{P}_Z|$ la taille asymptotique des ensembles $\mathcal{P}_z$. On remarque dans le décompte des contraintes, que le modèle *sommet-arc* augmente considérablement le nombre contraintes ($|V||E||Z|$) alors que la taille asymptotique de la formulation *arc-chemin* ne change pas.

À partir de l'expression du monoroutage sous sa version *sommet-arc*, on peut facilement adapter ce programme linéaire pour en faire une version "bi-routée". Là où l'on compte l'arité sortante des nœuds du réseau à l'aide de la variable $y_{e,z}$ dans l'équation 4.5, on va maintenant limiter à 1 l'arité entrante (équation 4.18) et sortante (équation 4.17), sauf pour les nœuds de départ (équation 4.19) et d'arrivée (équation 4.20) de la requête qui sont limités à 2.

Programme linéaire 7 (protection par 2 monoroutages disjoints *sommet-arc*)**.**

$$\forall e \in E,\ \sum_{z \in Z} x_{e,z} \leq W_f \times nb_f(e) \tag{4.13}$$

$$\forall e \in E,\ W_f \times nb_f(e) \leq c(e) \tag{4.14}$$

$$\forall v \in V, \forall z \in Z,\ \sum_{e^- \in \delta^-(v),\ e^+ \in \delta^+(v)} x_{e^-,z} - x_{e^+,z} = drop(v,z) - add(v,z) \tag{4.15}$$

$$\forall v \in V, \forall z \in Z,\ \forall e \in E,\ x_{e,z} \leq C \times y_{e,z} \tag{4.16}$$

$$\forall z = (s,t) \in Z,\ \forall v \in V \backslash \{s,t\},\ \sum_{e^+ \in \delta^+(v)} y_{e^+,z} \leq 1 \tag{4.17}$$

$$\forall z = (s,t) \in Z,\ \forall v \in V \backslash \{s,t\},\ \sum_{e^- \in \delta^-(v)} y_{e^-,z} \leq 1 \tag{4.18}$$

$$\forall z = (s,t) \in Z,\ \sum_{e^+ \in \delta^+(s)} y_{e^+,z} = 2 \tag{4.19}$$

$$\forall z = (s,t) \in Z,\ \sum_{e^- \in \delta^-(t)} y_{e^-,z} = 2 \tag{4.20}$$

$$\forall e \in E, \forall z \in Z,\ x_{e,z},\ y_{e,z},\ nb_f(e) \quad \textit{entiers} \tag{4.21}$$

$$\textit{Min} \sum_{e \in E} nb_f(e) \tag{4.22}$$

Algorithme 7 Reroutage de bout en bout.

Entrée: $G = (V, E)$, Z, $c(e)$, $\forall e \in E$
Sortie: Un ensemble de routes principales $\mathcal{R}(z)$, un ensemble de routes $\mathcal{R}(z, e)$ pour chaque cas de panne $e \in E, \forall z \in Z$
1: Calculer un multiflot avec G, Z et $c : e \rightarrow \mathbb{N}$
2: Sauver les chemins dans $\mathcal{R}(z)$
3: **pour tout** $e \in E$ **faire**
4: $G_e = (V, E \backslash \{e\})$
5: Soit $\mathcal{R}(z)_e$ l'ensemble des routes extraites de $\mathcal{R}(z)$ passant par e.
6: **pour tout** $e' \in E \backslash \{e\}$ **faire**
7: $c'(e') = c(e') - \sum_{p \in \mathcal{R}(z)_{e'}, p \notin \mathcal{R}(z)_e} size(p)$
8: **fin pour**
9: Soit Z_e le graphe des demandes associé à $\mathcal{R}(z)_e$.
10: Calculer un multiflot avec G_e, Z_e et $c' : e' \rightarrow \mathbb{N}$
11: Sauver les chemins dans $\mathcal{R}(z, e)$
12: **fin pour**

Le nombre de variables de cette formulation restent à $\mathcal{O}(|E||Z| + |E|)$ et le nombre de contraintes augmentent de $\mathcal{O}(|Z|)$, soit un total de $\mathcal{O}(|V||E||Z| + |V||Z| + |E||Z| + |E| + |Z|)$ contraintes.

Enfin, l'utilisation du monoroutage dans son modèle *arc-chemin* reste difficile à adapter pour la protection $1 : 1$, le nombre de contraintes d'exclusion augmente exponentiellement avec la taille des chemins candidats. Il devient alors difficile, en pratique, d'exploiter ce modèle. Il vaut alors mieux se tourner vers des algorithmes dédiés de calcul de k plus courts chemins disjoints [**Sch94, RS95**].

4.2.3. Algorithmes de calcul de protection $M : N$

La section 4.2.1 présente une technique de protection qui effectue un routage par cas de panne. En s'inspirant de cette méthode, on peut très facilement adapter cet algorithme pour élaborer deux algorithmes de protection qui, cette fois, ne mettent pas en jeu l'ensemble des connexions du réseau lorsqu'il s'agit de restaurer une panne.

4.2.3.1. *Protection par reroutage de bout en bout.* L'algorithme de *reroutage de bout en bout* calcule, pour chaque panne, un nouveau routage pour les routes principales concernées par une panne donnée. Ces routes principales sont retirées du réseau et un nouveau routage ne concernant que ces routes est effectué. De nouvelles routes de protection sont ainsi créées. Pour deux pannes différentes, ces routes de protection peuvent partager la capacité sur les arêtes du réseau.

L'algorithme 7 présente les étapes abrégées d'un tel algorithme. La boucle de la ligne 6 à 8 permet de calculer les nouvelles capacités du graphe résiduel G_e, c'est-à-dire le réseau avec toutes les routes non affectées par la panne en e. On extrait aussi en ligne 9 les demandes à router sur G_e. Il suffit de considérer chaque route passant par e et de lui associer une demande de même taille.

4.2.3.2. *Protection par reroutage autour de la panne.* L'algorithme de *reroutage autour de la panne* est très similaire à la précédente. Au lieu de rerouter les connexions affectées par la panne en e depuis la source et vers la destination de chaque route, on préfère créer une ou plusieurs

Algorithme 8 Reroutage autour de la panne.

Entrée: $G = (V, E)$, Z, $c(e)$, $\forall e \in E$
Sortie: Un ensemble de routes principales $\mathcal{R}(z)$, un ensemble de routes $\mathcal{R}(z, e)$ pour chaque cas de panne $e \in E, \forall z \in Z$
1: Calculer un multiflot avec G, Z et $c : e \rightarrow \mathbb{N}$
2: Sauver les chemins dans $\mathcal{R}(z)$
3: **pour tout** $e \in E$ **faire**
4: $G_e = (V, E \backslash \{e\})$
5: Soit $\mathcal{R}(z)_e$ l'ensemble des routes extraites de $\mathcal{R}(z)$ passant par e.
6: **pour tout** $e' \in E \backslash \{e\}$ **faire**
7: $c'(e') = c(e') - \sum_{p \in \mathcal{R}(z)_{e'}} size(p)$
8: **fin pour**
9: Si $e = (s, t)$, soit z_e la demande (s, t) de taille $\sum_{p \in Rz_e} size(p)$.
10: Calculer un multiflot avec G_e, $\{z_e\}$ et $c' : e' \rightarrow \mathbb{N}$
11: Sauver les chemins dans $\mathcal{R}(z, e)$
12: **fin pour**

routes qui vont contourner l'arête en panne. On conserve donc toutes les routes affectées par la panne en e : ces routes sont divisées en deux routes, l'une précédant e et l'autre suivant e. On cherche alors à allouer une nouvelle route qui relie les extrémités de e avec la capacité restante sur le réseau.

L'algorithme 8 diffère en ligne 7 où la capacité restante sur le réseau est retranchée de la capacité utilisée par toutes les routes principales. En ligne 9, une seule demande est considérée : celle qui contourne $e = (s, t)$ et qui reroute l'ensemble des chemins qui passent initialement par e.

Entre ces deux algorithmes, le partage entre les routes de protection n'est pas le même. Nous expliquons en section 4.3.2 les différents cas de partage possibles et nous reviendrons sur l'analyse de la performance de ces algorithmes, en terme de réutilisabilité des longueurs de partage.

Notons enfin que dans [**WSM02**] on considère une technique de protection à mi-chemin entre le reroutage de bout en bout et le reroutage autour de la panne. Il s'agit d'une protection par "sous-chemin" permettant au chemin interrompu en $e = (s, t)$ de repartir de s par un nouveau chemin rejoignant la destination de la requête z.

4.3. Un programme linéaire de protection $M : N$

Le modèle présenté dans [**BBG99**] présente un modèle de protection $M : N$ basé sur le partage des longueurs d'onde de protection. On cherche à y réaliser un routage assorti d'une protection complète et tolérant un cas de panne unique.

4.3.1. Modèle de Baroni, Bayvel et Gibbens

La particularité de l'approche est la suivante : au lieu d'aborder le problème en cherchant à construire la solution, les auteurs cherchent à énumérer les chemins possibles et pour ces ces chemins, les chemins de protection possible. On se rapproche alors du modèle de multiflot *arc-chemin* présenté en section 2.1.2 où l'on doit précalculer des chemins avec l'aide d'heuristiques comme expliqué en section 2.2. Dans [**BBG99**], à partir d'un ensemble de chemins très proche

des plus courts chemins, un programme linéaire est posé permettant de sélectionner parmi cet ensemble de chemins ceux qui donnent un ensemble de routes principales et de protection tout en cherchant une solution peu coûteuse en nombre de fibres.

Le programme linéaire posé est assez complexe. Nous présentons une version abrégée de ce programme linéaire (Programme linéaire 8) qui permettra, par la suite, d'introduire une méthode qui s'appuie fortement sur le modèle de [**BBG99**]. Nous adaptons aussi une partie des notations issues de [**BBG99**] pour permettre au lecteur de se rapporter aux notations présentes, notamment les notations E, Z et $\mathcal{P}_z^\mu$ qui font respectivement référence aux arêtes du graphe G, aux requêtes et à l'ensemble de chemins générés pour chaque requête $z \in Z$ et un paramètre μ contrôlant cette génération (cf. section 2.2). Notons aussi qu'à la place des arêtes $e \in E$, on utilisera la notation $j \in E$ pour se rapprocher des notations de [**BBG99**]. $\mathcal{F}_j$ est l'ensemble des chemins principaux passant par j, i.e. $\forall j \in E,\ \mathcal{F}_j = \{p | \exists z \in Z,\ p \in \mathcal{P}_z^\mu,\ j \in p\}$. Dans ce modèle on distingue l'ensemble des chemins principaux possibles des chemins de protection possibles. Ainsi, là ou l'on définit $\mathcal{P}_z^\mu$ pour les chemins principaux, on définit de même $\mathcal{R}_{p,j,b}$ comme l'ensemble des chemins de protection qui protègent le chemin p en cas de panne sur l'arête j (b contrôlant la taille de cet ensemble). Le programme linéaire 8 utilise deux types de variables différentes : $\delta^A_{p,z}$ qui vaut 0 ou 1 et qui décide du chemin utilisé pour la requête $z \in Z$, et la variable $\delta^R_{r,p,j}$ qui décide du chemin utilisé pour protéger le chemin p pour une panne $j \in E$ donnée. La notation $I(j \in p$ est une fonction indicatrice qui vaut 1 quand l'arc j fait partie du chemin p.

Programme linéaire 8 (adapté de Baroni, Bayvel, et Gibbens)**.**

$$\forall z \in Z,\ \sum_{p \in \mathcal{P}_z^\mu} \delta^A_{p,z} = 1 \tag{4.23}$$

$$\forall j \in E,\ \forall p \in \mathcal{F}_j,\ \sum_{r \in \mathcal{R}_{p,j,b}} \delta^R_{r,p,j} = \delta^A_{p,(s(p),d(p))} \tag{4.24}$$

$$\begin{aligned} \forall j \in E,\ \forall j' \neq j \in E,\ & \sum_{z \in Z} \sum_{p \in \mathcal{P}_z^\mu / p \notin \mathcal{F}_{j'}} \delta^A_{p,z}.I(j \in p) \\ & + \sum_{p \in \mathcal{F}_{j'}} \sum_{r \in \mathcal{R}_{p,j',b}} \delta^R_{r,p,j'}.I(j \in r) \leq W.f_j \end{aligned} \tag{4.25}$$

$$\forall j \in E,\ W.f_j \leq c(e) \tag{4.26}$$

$$\forall z \in Z,\ \forall p \in \mathcal{P}_z^\mu,\ \delta^A_{p,z},\quad \textit{entiers} \tag{4.27}$$

$$\forall p \in \mathcal{P}_z^\mu,\ \forall j \in E,\ \forall r \in \mathcal{R}_{p,j,b},\ \delta^R_{r,p,j},\quad \textit{entiers} \tag{4.28}$$

$$\textit{Min} \sum_{j \in E} f_j \tag{4.29}$$

Dans ce programme linéaire, l'équation 4.23 permet de choisir un chemin principal pour la requête $z \in Z$. L'équation 4.24 assure que l'on choisit un chemin de protection pour une panne donnée par rapport au chemin principal choisi. Ensuite, l'équation 4.25 assure que le calcul de consommation de la capacité disponible est admissible. En particulier, le modèle prend en compte le cas de partage de ressources explicité en section 1.3.4.3.

On note une particularité dans ce modèle : les requêtes $z \in Z$ sont forcément considérées comme des requêtes unitaires, ce qui diffère par rapport aux modèles de routage du flot de la section 2.1.2. Pour adapter cette méthode à la notation classique des données utilisées dans

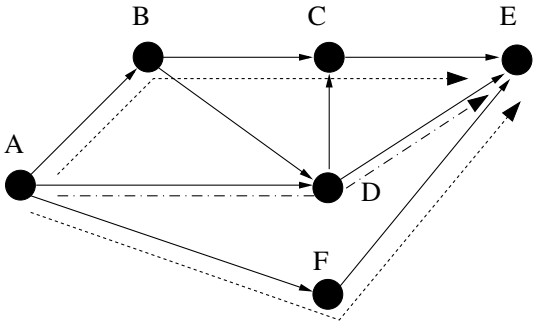


FIG. 4.2 – Partage de la capacité avec le même chemin de longueur d'onde : deux routes principales (pointillés courts) et un chemin de protection (pointillés discontinus)

cette thèse, il suffit alors de dupliquer chaque requête de taille $size(z)$ en autant de requêtes unitaires. Une autre différence entre le modèle de [**BBG99**] et celui qui est présenté ici, réside dans le fait que les auteurs se placent dans le cadre du dimensionnement de réseau. Ainsi, l'équation 4.26 n'existe pas dans leur modèle et est remplacée par $\forall j \in E, f_j \geq 1$ pour forcer l'utilisation d'au moins une fibre par arc. Ils considèrent en effet qu'il n'est pas raisonnable d'avoir un câble du réseau qui n'est pas du tout utilisé. Assez logiquement, le modèle étant fait pour du dimensionnement, ils cherchent à minimiser le nombre de longueurs d'onde utilisées, fonction objectif que nous conservons également et qui est similaire aux fonctions objectifs utilisées dans les sections 2.1.2 et 3.4.

La taille totale de ce programme linéaire est de l'ordre de $\mathcal{O}(|\mathcal{P}_Z||\mathcal{R}_Z||E| + |\mathcal{P}_Z||Z| + |E|)$ variables et $\mathcal{O}(|\mathcal{P}_Z||\mathcal{R}_Z||E| + |\mathcal{P}_Z||Z| + |\mathcal{P}_Z||E| + |E|^2 + |E| + |Z|)$ contraintes, où $|\mathcal{R}_Z|$ est la taille asymptotique des chemins de restauration et $|\mathcal{P}_Z|$ celle des chemins principaux. Cette taille augmente considérablement si l'on cherche à enrichir le modèle en prenant en compte le cas la conversion des longueurs d'onde : on introduit une nouvelle dimension aux variables désignant quelle longueur d'onde est utilisée sur le réseau. Dans un tel modèle on peut alors faire de l'affectation de longueur d'onde, sans pour autant nécessiter de fonctionnalité de conversion en longueurs d'onde dans les nœuds. Dans [**BBG99**] on montre alors que la taille du problème est multipliée par W, voire par W^2 pour certains termes.

4.3.2. Partage des ressources de protection

Le cas classique de partage des ressources de protection est présenté en figure 4.2. Lorsque deux chemins principaux sont disjoints, et dans l'hypothèse d'une seule panne simultanée, il ne peut y avoir qu'un seul chemin principal protégé par le chemin de secours au même moment. Dans ce cas, une seule longueur d'onde est réservée sur le chemin $(A \rightarrow D \rightarrow E)$ pour protéger les chemins $(A \rightarrow B \rightarrow D \rightarrow E)$ et $(A \rightarrow F \rightarrow E)$.

La figure 4.3 présente un cas de partage moins fréquemment pris en compte dans la littérature. Il s'agit de partager une ressource entre le chemin principal et le chemin de protection pour un cas de panne donné. Lorsque l'arc j est en panne, la route (A, B, C, E) tombe en panne et bascule sur la route de protection (A, D, E). Au niveau de l'arc k, on peut alors utiliser la même longueur d'onde, au lieu d'en allouer deux différentes. Évidemment, cela suppose que le nœud C soit reconfiguré pour envoyer sur une même longueur d'onde de l'arc (C, E) une longueur d'onde provenant de B ou de D.

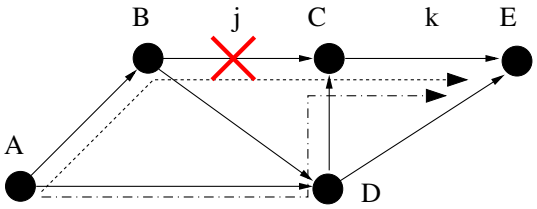


FIG. 4.3 – Partage de la capacité sur un lien k entre la route principale (pointillés courts) et un chemin de protection (pointillés discontinus) en cas de panne sur l'arc j

Ce deuxième cas de partage n'est pas pris en compte dans [**BBG99**] et nous proposons, dans la suite, de l'inclure dans notre modèle puisqu'il permet de réduire considérablement les ressources nécessaires à la protection totale du trafic. Notons aussi qu'en section 4.2.3 le *reroutage de bout en bout* et le *reroutage autour de la panne* utilise cette méthode de partage puisque, pour chaque cas de panne, les chemins principaux sont ajoutés à la capacité résiduelle du réseau.

4.3.3. Limitations du modèle

L'adaptation du modèle de Baroni, Bayvel et Gibbens est un programme linéaire massivement $0/1$, c'est-à-dire utilisant de nombreuses variables binaires. De plus, le modèle introduit une dépendance forte entre les variables des chemins principaux et les variables des chemins de protection. Dans l'hypothèse d'utilisation de technique du type *branch and bound*, l'exploration des possibilités est d'autant plus difficile. Les auteurs mentionnent déjà ces limitations et proposent des résultats expérimentaux sur des réseaux de petite taille. Si l'on ajoute sur le modèle les variables d'affectation des longueurs d'onde, il devient quasiment impossible à résoudre.

Par ailleurs, le modèle présenté propose d'attribuer une variable pour chaque requête unitaire du modèle. Cela permet d'étendre ce modèle à la problématique d'affectation de longueurs d'onde, ce qui est proposé dans [**BBG99**]. Cependant, si on laisse de côté cette problématique, nous pensons qu'il est préférable d'associer à chaque chemin une certaine quantité de longueurs d'onde, ce qui se rapproche des modèles de multiflot *arc-chemin* qui associent à un chemin une certaine quantité de flot. En fait, les algorithmes de programmation linéaire résolvant des multiflots fonctionnent plus efficacement sur des variables du type $[0, \ldots, \gamma]$.

Dans la suite de ce chapitre nous proposons d'adapter ce modèle pour le rendre plus efficace dans la pratique. Nous présentons comment reformuler les contraintes de protection de façon à pouvoir relâcher la contrainte d'intégrité des variables du modèle et se contenter d'une résolution fractionnaire.

4.4. Modèle mixte

À partir du modèle de Baroni, Bayvel et Gibbens, présenté en section 4.3.1 nous proposons de construire un modèle mixte, c'est-à-dire mêlant l'expression du multiflot *sommet-arc* et *arc-chemin*. Ce modèle mixte permettra par la suite de résoudre une relaxation du problème et de proposer une méthode permettant d'extraire une solution entière[1].

[1]Ce travail est en cours de publication [**LSV05**]

4.4.1. Routes principales

La première partie du programme linéaire reprend directement l'équation 4.23 mais en considérant cette fois qu'une requête n'est pas unitaire. Ainsi, un chemin $p \in \mathcal{P}_z^\mu$ pourra router un ensemble de longueurs d'onde pour satisfaire les $size(z)$ longueurs d'onde de la demande z.

Programme linéaire 9 (Equations pour les routes principales)**.**

$$\forall z \in Z,\ \sum_{p \in \mathcal{P}_z^\mu} \delta_{p,z}^A \quad = \quad size(z), \tag{4.30}$$

$$\forall z \in Z,\ \forall p \in \mathcal{P}_z^\mu,\ \delta_{p,z}^A \qquad \textit{entier} \tag{4.31}$$

4.4.2. Flot de protection

Pour chaque route principale p, on doit calculer une route de protection et ceci pour chaque cas de panne potentielle du câble $j \in p$. Dans le modèle de la section 4.3.1, une variable de décision est créée pour chaque chemin de protection potentiel. Ce modèle nécessite alors de générer un nouvel ensemble de routes, $\mathcal{R}_{p,j,b}$ de manière similaire à $\mathcal{P}_z^\mu$. Pour éviter cet écueil, nous introduisons ici le modèle de multiflot *sommet-arc* pour modéliser le flot de routage autour de l'arc j. Nous définissons alors les routes de protection comme un flot évitant l'arc j et dont la taille est précisément égale au flot principal passant par j (cette valeur est donc reliée aux variables $\delta_{p,z}^A$. Évidemment, nous devons donc définir un flot par panne potentielle le long de chaque chemin $p \in \mathcal{P}_z^\mu$.

De manière plus formelle, nous définissons un multiflot sur le graphe $\mathcal{G}P_{z,j} = \mathcal{G}P_z \backslash \{j\}$ qui est le flot de protection lorsque j est en panne. $\mathcal{G}P_z$ est défini dans l'équation 4.32 comme un sous graphe de G ce qui permet de réduire le nombre de variables à inclure dans l'expression du multiflot. Ce sous graphe sera le support pour router la quantité de flot des chemins principaux utilisés. Il est construit à partir des routes principales de $\mathcal{P}_z^\mu$. La définition d'un flot sur un sous graphe de G permet de définir, comme dans le modèle de [**BBG99**] que les chemins de protection n'utilisent qu'une partie du réseau, focalisée autour des plus courts chemins. Dans [**BBG99**] on utilise $\mathcal{R}_{p,j,b}$, un ensemble de routes de protection dont on contrôle la distance par rapport au plus court chemin à l'aide de b. Dans notre modèle, étant donné que la définition du flot de protection pourrait couvrir potentiellement l'intégralité du graphe, il nous faut extraire un sous graphe candidat au support du flot. Pour cela, nous utilisons les caractéristiques des chemins principaux, qui sont contrôlés par le paramètre μ. $\mathcal{G}P_z$ est donc défini comme le sous graphe issu de la couverture de G par des chemins principaux candidats $p \in \mathcal{P}_z^\mu$ (équation 4.32).

$$\forall z \in Z,\ \mathcal{G}P_z = \{e \in E / \ \exists p \in \mathcal{P}_z^\mu,\ e \in p\}, \tag{4.32}$$

$$\forall z \in Z,\ \forall j \in E,\ \mathcal{G}P_{z,j} = \mathcal{G}P_z \backslash \{j\}. \tag{4.33}$$

Le programme linéaire 10 présente les équations classiques de conservation du flot (équation 4.34), d'introduction du flot au nœuds sources (équation 4.35), de sortie du flot aux nœuds destinations (équation 4.36).

Programme linéaire 10 (Equations de protection)**.**

$$\forall j \in E,\ \forall z = (s,t) \in Z, \forall v \in V,\ v \notin \{s,t\},$$

$$\sum_{e \in \delta^-(v) \cap \mathcal{G}P_{z,j}} f^j_{e,z} = \sum_{e \in \delta^+(v) \cap \mathcal{G}P_{z,j}} f^j_{e,z}, \tag{4.34}$$

$$\forall j \in E,\ \forall z = (s,t) \in Z \quad \sum_{e \in \delta^+(s) \cap \mathcal{G}P_{z,j}} f^j_{e,z} = \sum_{j \in p} \delta^A_{p,z}, \tag{4.35}$$

$$\sum_{e \in \delta^-(t) \cap \mathcal{G}P_{z,j}} f^j_{e,z} = \sum_{j \in p} \delta^A_{p,z}, \tag{4.36}$$

$$\forall e \in \mathcal{G}P_{z,j},\ f^j_{e,z} \geq 0,\ \textit{entier} \tag{4.37}$$

4.4.3. Exemple d'un jeu de variables pour les programmes linéaires 9 et 10

L'exemple de la figure 4.4 présente les contraintes présentées précédemment. Pour une requête $z = (A, E)$, nous montrons 3 routes possibles : p_1, p_2, ou p_3. Sur le schéma du bas, nous montrons un cas de panne sur le câble AB affectant le chemin principal p_1. On introduit alors en A un flot de taille $\delta^A_{p_1,z}$ qui sort en E et qui utilise le graphe $\mathcal{G}P_{z,AB}$. $\mathcal{G}P_{z,AB}$ résulte du graphe $\mathcal{G}P_z$ privé de l'arc en panne, (AB). La figure donne une des équations de conservation du flot, au niveau du nœud D et montre bien que le flot $f^{AB}_{DE,z}$ n'y est pas comptabilisé.

4.4.4. Contraintes de capacité

Nous présentons dans cette section les contraintes de capacité qui vont garantir que le nombre de longueurs d'onde allouées, à la fois pour les chemins principaux et les chemins de protection ne dépasseront pas $c(e),\ \forall e \in E$ la capacité d'un câble e compté en nombre de longueurs d'onde. Les variables $f_{e,z}$ représentent le nombre de longueurs d'onde de protection nécessaires, quel que soit le cas de panne considérée. Nous introduisons ces variables pour faciliter la compréhension du modèle. Nous montrons en section 4.5 une réécriture du modèle qui évite d'employer ces variables. Comme la capacité nécessaire à la protection est partagée entre tous les cas de panne, $f_{e,z}$ représente le flot de protection "maximum" sur l'arc e, i.e. $\forall z \in Z,\ \forall e \in E, \forall j \in E, f^j_{e,z} < f_{e,z}$. Cependant, cette contrainte n'est pas suffisante pour modéliser le partage des ressources de protection décrit en section 4.3.2. En effet, sur l'arc k de la figure 4.3, la variable $f_{e,z}$ doit valoir 0 puisqu'aucune capacité de protection n'est nécessaire. Ce cas est pris en compte dans l'équation 4.38, si $\rho = 1$. Lorsque $\rho = 0$, il n'y a pas de partage entre une route principale et une route de protection qui ont un arc commun.

Programme linéaire 11 (Equations de capacité)**.**

$$\forall z \in Z,\ \forall (e,j) \in E^2,\ f^j_{e,z} - \rho \sum_{p \in \mathcal{P}^\mu_z,\ j \in p,\ e \in p} \delta^A_{p,z} \leq f_{e,z} \tag{4.38}$$

$$\forall e \in E,\ \sum_{z \in Z} \Big(\sum_{p \in \mathcal{P}^\mu_z / e \in p} \delta^A_{p,z} + f_{e,z} \Big) \leq c(e) \tag{4.39}$$

4.4.5. Fonction objectif

Dans ce modèle, nous utilisons une fonction objectif similaire à [**BBG99**], quantifiant la charge du réseau. Nous choisissons de minimiser le nombre de fibres allouées sur le réseau,

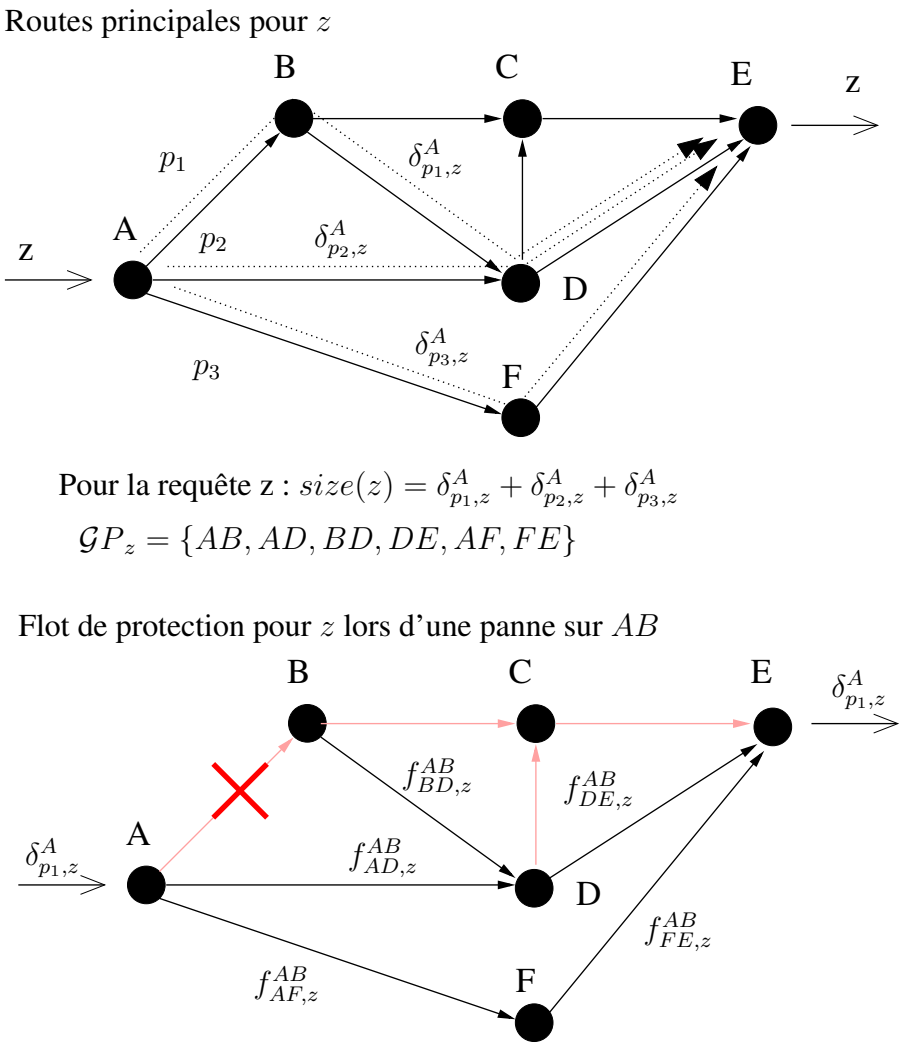


FIG. 4.4 – Variables du programme linéaire pour la requête $z = (A, E)$

comme dans le chapitre 2 traitant du routage. Là encore, un effet de "saut" s'observe à cause de la multiplicité W. Pour calculer cette fonction objectif, nous introduisons $nb_f(e)$ comme le nombre de fibres allouées sur l'arc e. On a alors la fin du programme linéaire du modèle suivant :

Programme linéaire 12 (Equations pour la fonction objectif)**.**

$$\forall e \in E, \sum_{z \in Z} \Big(\sum_{e \in p \in \mathcal{P}^{\mu}_z} \delta^A_{p,z} + f_{e,z} \Big) \quad \leq \quad W.nb_f(e) \tag{4.40}$$

$$Min \sum_{e \in E} nb_f(e) \tag{4.41}$$

Algorithme 9 Arrondi aléatoire des chemins principaux et de protection

Entrée: $G = (V, E)$, Z, $\mathcal{P}_z^\mu$, $\delta_{p,z}^A$, $f_{u,z}$, $\mathcal{G}P_{z,j}$
Sortie: Un ensemble de routes principales $\mathcal{R}(z)$, de routes de protection $\mathcal{R}(z, j)$

1: **pour tout** $z = (s, t) \in Z$ **faire**
2: **pour tout** longueur d'onde de z **faire**
3: Choisir un chemin $p \in \mathcal{P}_z^\mu$ avec probabilité $\delta_{p,z}^A / (\sum_{p' \in \mathcal{P}_z^\mu} \delta_{p',z}^A)$
4: Sauver p dans $\mathcal{R}(z)$.
5: **pour tout** $j \in p$ **faire**
6: Soit $v = s$. Créer un chemin vide p'
7: {Marche aléatoire de s à t dans $G(V, \mathcal{G}P_{z,j})$}
8: **tant que** $v \neq t$ **faire**
9: Choisir e dans $\delta^+(v) \cap \mathcal{G}P_{z,j}$ avec probabilité $f_{e,z}^j / (\sum_{u \in \delta^+(v) \cap \mathcal{G}P_{z,j}} f_{u,z}^j)$.
10: Concaténer e à p' ; $v = head(e)$.
11: **fin tant que**
12: Sauver p' dans $\mathcal{R}(z, j)$.
13: **fin pour**
14: **fin pour**
15: **fin pour**

Au total, le modèle mixte comporte $\mathcal{O}(|\mathcal{P}_Z||Z| + |Z||E|^2 + |E||Z| + |E|)$ variables et $|\mathcal{O}(|V||E||Z| + |Z||E|^2 + |Z||\mathcal{P}_Z| + |E||Z| + |E| + |Z|)$. Par rapport au modèle de [**BBG99**], le nombre de variables ne dépend plus du nombre de chemins de protection et le nombre de contraintes augmente significativement.

4.5. Relaxation du modèle et arrondi aléatoire

Le modèle que nous venons de présenter en section 4.4 est construit pour faciliter la relaxation de l'intégrité des variables. Alors que dans [**BBG99**] les variables $0/1$ des routes principales sont liées à des variables $0/1$ pour les routes de protection, notre modèle introduit des variables appartenant à l'ensemble $\{0 \dots \gamma\}$, que ce soit pour les chemins principaux ou pour le flot de protection. Par ailleurs, nous montrons qu'à l'aide d'une technique adaptée d'arrondi aléatoire, nous pouvons extraire d'une solution fractionnaire une solution entière de bonne qualité asymptotiquement presque sûrement.

4.5.1. Arrondi aléatoire du modèle de la section 4.4

L'algorithme d'arrondi aléatoire présenté ici est basé sur l'algorithme 2 de la section 2.3.1 qui décrit le processus de marche aléatoire pour un multiflot fractionnaire. Le principe général est le suivant : on arrondit en premier les chemins principaux (ligne 3), puis pour chaque chemin principal ainsi créé on réalise une marche aléatoire sur le flot de protection (ligne 8 à 11). Cet algorithme est polynomial et réalise un arrondi de l'ensemble des variables du programme linéaire (les variables f_j s'en déduisent). Notons enfin que l'algorithme introduit une dépendance entre le processus de marche aléatoire pour la protection et le tirage des chemins principaux. Ce problème de dépendance est explicité par la suite dans la partie consacrée aux garanties théoriques d'un tel algorithme.

4.5.2. Qualité de l'arrondi aléatoire pour un multiflot

Le résultat suivant s'inspire d'un théorème similaire de [**RT87, Rag94**]. Il démontre la qualité de la solution obtenue pour l'algorithme 9 et reste valable, dans un cadre plus général, pour l'algorithme classique d'arrondi d'un multiflot, présenté en section 2.3.1 dans l'algorithme 2. Nous donnons les détails spécifiques de la preuve au cas particulier du flot principal et de protection dans la section 4.5.3.

Proposition 2. *Si sur chaque arc e la capacité associée $c(e)$ est telle que $c(e) \geq 12 \ln |E|$ Alors, avec probabilité $1 - 1/|E|$ l'algorithme 2 conduit à une solution entière où toutes les contraintes de capacités sont respectées.*

La démonstration du résultat utilise les bornes de Chernoff [**Che52**]. En particulier, nous utilisons la formulation de [**AV79**] suivante :

Théorème 1 (Chernoff)**.** *Soit $X_1, \ldots, X_n$ un ensemble de variables aléatoire indépendantes telles que $1 \leq i \leq n$, $\Pr[X_i = 1] = p_i$ où $0 \leq p_i \leq 1$. Alors pour $X = \sum_{i=1}^{n} X_i$, $\mathbb{E}[X] = \sum_{i=1}^{n} p_i$,*

$$\Pr[X > (1+\varepsilon)\mathbb{E}[X]] \leq \left(\frac{e^{\varepsilon}}{(1+\varepsilon)^{(1+\varepsilon)}} \right)^{\mathbb{E}[X]} \text{ avec } \varepsilon > 0, \tag{4.42}$$

et

$$\Pr[X > (1+\delta)\mathbb{E}[X]] \leq \exp\left(-\frac{\mathbb{E}[X]\delta^2}{3} \right) \text{ avec } 0 \leq \delta \leq 1. \tag{4.43}$$

Pour la preuve de la proposition 2, il faut noter que l'évènement "la marche aléatoire pour une commodité i traversant e" est un tirage de Bernouilli. Cette observation est formulée par le lemme suivant, si $\hat{f}_i$ représente le flot fractionnaire de la commodité i :

Lemme 2. *La probabilité que la marche aléatoire de la commodité i traverse un arc e est égale à $\hat{f}_i(e)$.*

Démonstration. Ce lemme se prouve aisément par induction sur chaque nœud du graphe en remarquant que pour chaque nœud v du réseau on tire un arc avec une probabilité

$$f_i(e) / \sum_{e \in \delta^+ v} f_i(e).$$

□

Preuve de la proposition 2 : L'algorithme 2 produit une solution entière ; la seule possibilité pour que la solution ne soit pas acceptable est qu'elle viole les contraintes de capacité sur certains arcs. Après le processus d'arrondi aléatoire, il faut noter que le flot sur chaque arc, est une somme de tirages de Bernouilli indépendants (lemme 2). Soit e un arc, et soit X le flot de cet arc après le processus d'arrondi aléatoire. Calculons la quantité $\Pr[X > c(e)]$. On notera $\mathbb{E}[X]$ l'espérance mathématique de X. Nous considérons alors les deux cas suivants :

Cas $\mathbb{E}[X] \geq c(e)/2$:

On applique alors le théorème 1 et l'équation 4.43 donne alors,

$$\begin{aligned}\Pr\left[X > (1+\delta)\mathbb{E}[X]\right] &\leq \exp\left(-\frac{\delta^2\mathbb{E}[X]}{3}\right)\\ &\leq \exp\left(-\frac{\delta^2 c(e)}{6}\right)\\ &\leq \frac{1}{|E|^2},\end{aligned}$$

en choisissant $\delta = \sqrt{(12\ln|E|)/c(e)}$.

Comme $(1-\varepsilon)c(e) \geq \mathbb{E}[X]$ et $1/(1-\varepsilon) \geq (1+\delta)$ on obtient $\Pr[X > c(e)] \leq \Pr[X > (1+\delta)\mathbb{E}[X]] \leq \frac{1}{|E|^2}$.

Cas $\mathbb{E}[X] \leq c(e)/2$ *:* Nous appliquons cette fois l'équation 4.42 du théorème 1 et, en prenant $\varepsilon = 1$, on obtient :

$$\begin{aligned}\Pr[X > 2\mathbb{E}[X]] &\leq \left(\frac{e}{4}\right)^{\mathbb{E}[X]}\\ &\leq \exp\left(12(1-\ln 4)\ln|E|\right)\\ &\leq \frac{1}{|E|^{12(\ln 4-1)}}\end{aligned}$$

donc $\Pr[X > c(e)] \leq \frac{1}{|E|^2}$

Ainsi, dans tous les cas, on majore $\Pr[X > c(e)]$ par une quantité bornée par $\frac{1}{|E|^2}$. Etant donné que ce calcul est valable pour tous les arcs du réseau, la probabilité qu'un de ces évènements se produise est égale à la multiplication des probabilités indépendantes élémentaires. Au total, on obtient donc une probabilité de dépassement pour un arc quelconque de $\frac{1}{|E|}$. □

Cependant, cette garantie théorique n'est pas directement exploitable pour notre problème. On souhaite adapter cette borne pour que la capacité ne dépasse pas $(1-\epsilon)c(e)$. En réduisant $c(e)$ à $(1-\varepsilon)c(e)$ avec $0 < \varepsilon \leq (\sqrt{5}-1)/2$ nous forçons le flot fractionnaire à être légèrement au dessous de la capacité réelle ce qui permettra au flot arrondi d'être, avec une meilleure probabilité, au dessous de la capacité. Nous présentons alors un deuxième résultat qui quantifie la qualité de cette approche :

Proposition 3. *L'algorithme 2 produit une solution respectant les contraintes de capacité dont le flot total est au moins* $F(1-\varepsilon^2)$ *avec une probabilité* $2\exp(-0,38\varepsilon^2 F)$*, où* F *est le flot total optimum dans la solution optimal entière.*

Démonstration. Notons tout d'abord que le flot total $\hat{F}$ de la solution fractionnaire est telle que $\hat{F} > (1-\varepsilon)F$ (car $F = \hat{F}$). On en déduit donc que l'espérance du flot total arrondi $\hat{F}$ est tel que $\mathbb{E}[\hat{F}] > (1-\varepsilon)F$. De plus, le flot total $\hat{F}$ est aussi une somme de tirages de Bernouilli indépendants : on peut donc utiliser l'équation 4.43 du théorème 1 et l'on obtient,

$$\Pr[\hat{F} > (1-\varepsilon^2)F] < \exp\left(-\frac{\varepsilon^2(1-\varepsilon)F}{3}\right)$$

ce qui conclut la preuve. □

4.5.3. Qualité de l'arrondi aléatoire pour l'algorithme 9

La proposition 2 s'adapte à l'algorithme 9 assez simplement. Les seules équations mises en jeu dans la probabilité de dépassement de la capacité sont les équations 4.38 et 4.39. Pour faciliter la démonstration nous réécrivons ces équations sous une forme plus ramassée. Le lemme suivant peut facilement être vérifié par le lecteur :

Lemme 3. *Les contraintes 4.38 et 4.39 peuvent être réécrites en une seule contrainte :*

$$\forall (e,j) \in E^2, \sum_{z \in Z} \Big(\sum_{p \in \mathcal{P}_z^{\mu} / e \in p, j \notin p} \delta_{p,z}^{A} + f_{e,z}^{j} \Big) \quad \leq \quad c(e) \tag{4.44}$$

Proposition 4. *Si sur chaque arc e la capacité associée $c(e)$ est telle que $c(e) \geq 12 \ln |E|$ Alors, avec probabilité $1 - 1/|E|$ l'algorithme 9 conduit à une solution entière où toutes les contraintes de capacités sont respectées.*

Démonstration. La preuve est similaire à la preuve de la proposition 2. L'équation 4.44 est une somme de tirages de Bernoulli indépendants puisque les étapes de la ligne 3 et 9 de la marche aléatoire sont des tirages de Bernoulli. Si l'on considère deux exécutions de la boucle de la ligne 2, les tirages sont indépendants. Pendant l'exécution d'une boucle, pour une même requête z, les tirages de l'étape de la ligne 9 pendant l'exécution de la boucle de la ligne 8 ne sont jamais sommés entre eux, car ils concernent à chaque fois deux arcs distincts. Il reste donc à vérifier s'il peut y avoir la somme d'un tirage de la ligne 3 avec un tirage de la ligne 9 dans l'équation 4.44. Si ces deux quantités sont sommées, alors on est en train d'additionner une longueur d'onde principale et une longueur d'onde de protection pour une même requête z et c'est un cas de partage présenté en section 4.3.2. Ceci ne peut donc pas se produire : en pratique, dans l'équation 4.44 on évite de sommer ces deux quantités grâce à la contrainte $j \notin p$. □

4.5.4. Programme linéaire final

Suite à la présentation de l'algorithme 9, nous pouvons alors présenter l'intégralité du programme linéaire à mettre en œuvre, en tenant compte des capacités modifiées.

Cette version finale du programme linéaire mixte comporte $\mathcal{O}(|\mathcal{P}_Z||Z| + |Z||E|^2 + |E|)$ variables et $|\mathcal{O}(|V||E||Z| + |E|^2 + |Z||\mathcal{P}_Z| + |E||Z| + |E| + |Z|)$. Le nombre de contraintes et de variables a pu être légèrement réduit.

Programme linéaire 13 (Calcul des routes principales et de protection pour l'algorithme 9)**.**

$$\forall z \in Z,\ \sum_{p \in \mathcal{P}_z^\mu} \delta_{p,z}^A = size(z), \tag{4.45}$$

$$\forall z \in Z,\ \forall p \in \mathcal{P}_z^\mu,\ \delta_{p,z}^A \quad \textit{entier} \tag{4.46}$$

$$\forall j \in E,\ \forall z = (s,t) \in Z, \forall v \in \mathbb{N},\ v \notin \{s,t\},$$
$$\sum_{e \in \delta^-(v) \cap \mathcal{G}P_{z,j}} f_{e,z}^j = \sum_{e \in \delta^+(v) \cap \mathcal{G}P_{z,j}} f_{e,z}^j, \tag{4.47}$$

$$\forall j \in E,\ \forall z = (s,t) \in Z \sum_{e \in \delta^+(s) \cap \mathcal{G}P_{z,j}} f_{e,z}^j = \sum_{j \in p} \delta_{p,z}^A, \tag{4.48}$$

$$\sum_{e \in \delta^-(t) \cap \mathcal{G}P_{z,j}} f_{e,z}^j = \sum_{j \in p} \delta_{p,z}^A, \tag{4.49}$$

$$\forall e \in \mathcal{G}P_{z,j},\ f_{e,z}^j \geq 0,\ \textit{entiers} \tag{4.50}$$

$$\forall (e,j) \in E^2, \sum_{z \in Z} \Big(\sum_{p \in \mathcal{P}_z^\mu / e \in p, j \notin p} \delta_{p,z}^A + f_{e,z}^j \Big) \leq c(e) \tag{4.51}$$

$$\forall e \in E, \sum_{z \in Z} \Big(\sum_{e \in p \in \mathcal{P}_z^\mu} \delta_{p,z}^A + f_{e,z} \Big) \leq W.nb_f(e) \tag{4.52}$$

$$\textit{Min} \sum_{e \in E} nb_f(e) \tag{4.53}$$

4.6. Résultats expérimentaux

Nos expérimentations comparent notre implémentation du modèle de [**BBG99**] adapté en section 4.3.1 (noté *BBG* pour Baroni, Bayvel et Gibbens) et l'algorithme de programmation linéaire mixte 13 suivi de l'algorithme 9 d'arrondi aléatoire (noté *LSV* pour Lalande, Syska et Verhoeven). Nous présentons également la valeur obtenue pour le reroutage global, présenté en section 4.2.1 et noté *RG*.

Ces trois algorithmes ont été implémentés dans MASCOPT et interfacé à CPLEX SOLVER™ à la fois pour la résolution en entier de *BBG* et *RG* ou pour la résolution fractionnaire de *LSV*.

4.6.1. Choix expérimental de la taille des ensembles de chemins

Il faut noter que la génération des ensembles $\mathcal{P}_z^\mu$ n'est pas la même pour chaque algorithme, ceci pour des raisons purement pratiques : la résolution de *BBG* s'effectue à l'aide de la méthode *Mixed Integer Programming* de CPLEX SOLVER™ et nécessite un long processus de *branch and bound*. Si l'ensemble $\mathcal{P}_z^\mu$ est trop grand, il devient très vite impossible de résoudre des instances de grande taille. Cette limitation était déjà pointée dans [**BBG99**]. Dans le cas de l'algorithme *LSV*, on peut au contraire générer un ensemble $\mathcal{P}_z^\mu$ très large, puisque la résolution est faite par un algorithme du type *simplexe*.

D'autre part, suivant les expérimentations, le réglages de $\mathcal{P}_Z^\mu$ diffère pour des questions d'ordre pratique. Les résultats de la table 4.1 n'impliquent qu'un seul calcul par réseau, pour chaque algorithme. L'ensemble $\mathcal{P}_Z^\mu$ utilisé est donc assez large. En ce qui concerne la figure 4.5, on observe que le nombre de longueurs d'onde alloué, pour $W = 50$ est bien plus élevé que dans la table 4.1 : pour tracer la courbe en un temps raisonnable, la taille de $\mathcal{P}_Z^\mu$ est ici réduite, ce qui dégrade la qualité de la solution.

	W	Fibres		Longeurs d'onde			Temps moyen	
Réseau		*BBG*	*LSV*	*BBG*	*LSV*	*RG*	*BBG*	*LSV*
NSFNET	36	42	42	1114	884	1021	1h	2m
R_1	50	68	71	2435	1987	2675	20h	10m
R_3	20	49	52	930	733	900	2h	6m
EU	50	$\times$	517	$\times$	21945	33554	$\times$	30m

TAB. 4.1 – Nombre total de fibres et de longueurs d'onde avec *LSV* et *BBG*

4.6.2. Comparaison des algorithmes *BBG* et *LSV*

La table 4.1 présente les résultats obtenus sur divers réseaux détaillés en annexe A.2. Pour chaque instance, nous précisons la multiplicité utilisée W et nous comparons le nombre de fibres et de longueurs d'onde allouées par chaque algorithme. On observe qu'en général le nombre de fibres est légèrement moins bon pour l'algorithme *LSV* par rapport à *BBG*. Ceci est dû au processus d'arrondi aléatoire qui produit un "saut" au niveau des fibres allouées lorsque le nombre de longueurs d'onde dépasse un multiple de W. En effet, le programme linéaire fractionnaire peut très bien réussir à calculer un nombre de longueur d'onde multiple de W (et dans la version du théorème 4, un multiple de $W.(1-\varepsilon)$). Lors du processus d'arrondi, il suffit que le nombre de longueurs d'onde soit de $k.W+1$ pour forcer l'allocation de $(k+1)$ longueurs d'onde.

Cependant, lorsque l'on compare le nombre de longueurs d'onde allouées, on constate que *LSV* permet d'en économiser un quantité non négligeable. On peut expliquer cela par le fait que *LSV* prend en compte un cas de partage supplémentaire, par rapport à *BBG* (cf. section 4.3.2). D'autre part, l'augmentation de la taille de $\mathcal{P}z^{\mu}$ permet d'introduire une plus grande flexibilité des chemins et donc, d'en choisir des meilleurs en terme de consommation de ressources. Enfin, on peut supposer que l'expression de la protection en tant que multiflot permet encore une plus grande flexibilité dans la répartition des chemins, par rapport à un modèle *arc-chemin*, limité par la taille de l'ensemble des chemins considérés.

Par rapport à la borne supérieure du reroutage globale noté *RG*, nous ne présentons que le nombre de longueurs d'onde obtenues : l'implémentation de *RG* a été faite dans [**Mos04**] avec une fonction objectif spécifique cherchant à minimiser le nombre de longueurs d'onde (et non pas le nombre de fibres). On observe alors que *LSV* est toujours au dessous de cette borne supérieure alors que *BBG* la dépasse presque à chaque fois.

4.6.3. Évolution avec la multiplicité

Les figures 4.5 et 4.6 présentent l'évolution des fibres et longueurs d'onde allouées en fonction de la multiplicité W. Nous observons le phénomène déjà mentionné dans [**BBG99**] à savoir que le nombre de ressources à allouer dépend fortement de la connectivité du réseau et qu'en augmentant W, ce qui ajoute de la flexibilité à l'instance, on réduit plus que linéairement le nombre de ressources à utiliser.

On observe sur la figure 4.6 un effet de seuil pour l'algorithme *BBG* qui n'arrive plus à tirer partie de l'augmentation de W à partir de $W=44$. Le nombre de fibres allouées par *LSV* décroît presque linéairement jusqu'à $W=50$ où l'on observe encore un seuil.

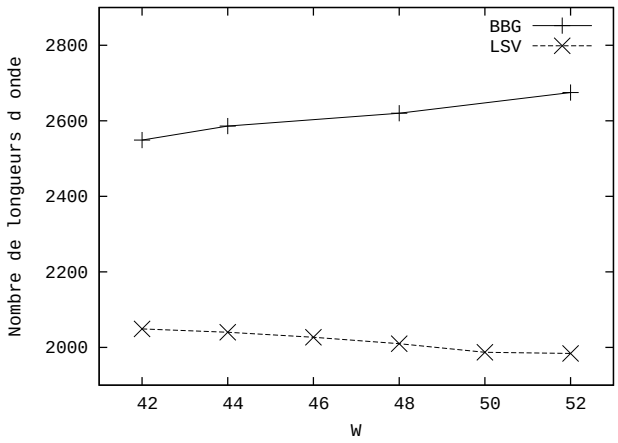


FIG. 4.5 – Nombre total de longueurs d'onde allouées en fonction de W pour R_1

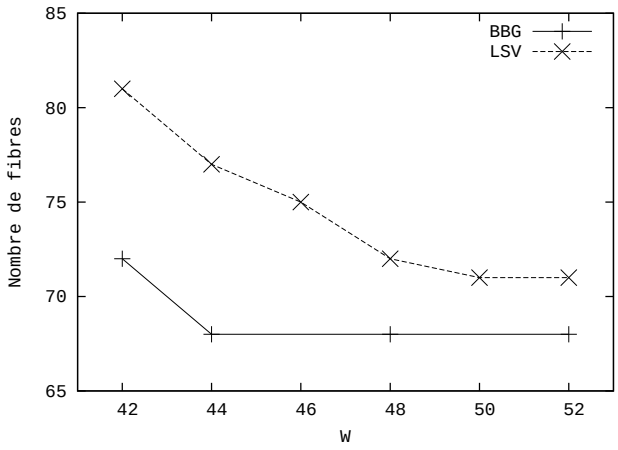


FIG. 4.6 – Nombre total de fibres allouées en fonction de W pour R_1

4.6.4. Interprétation des résultats

Les expérimentations précédentes montrent que l'algorithme *LSV* permet d'avoir des chemins plus chargés que les chemins créés par *BBG*. *LSV* trouve des chemins plus courts ce qui permet d'économiser des longueurs d'onde, quitte à allouer de nombreuses fibres sur ces chemins. On obtient alors un réseau très chargé sur certains câbles (les arcs du "centre" du réseau), alors que d'autres câbles périphériques ne le sont pas.

Ce résultat peut entrer en contradiction avec des travaux où l'objectif est de minimiser la charge maximum du réseau. En général, ces travaux traitent des problèmes plus proches de la conception que de l'allocation de ressources : on minimise la charge maximum pour répartir la capacité de manière uniforme sur le réseau (sinon, on allouerait tout le temps le long des plus courts chemins). Or, dans nos hypothèses, on ne tient pas compte de cette charge maximum puisque les fibres du réseau sont déjà existantes et que la communication au travers des fibres optiques n'est pas gêné par une saturation sur un lien du réseau.

Conclusion

Ce chapitre a permis de présenter la problématique de la protection dans les réseaux optiques. Après avoir présenté un état de l'art des différents types de protection et de certains algorithmes permettant de les calculer, nous avons montré comment adapter les modèles de programmation linéaire du routage pour la protection $1:1$. Nous avons présenté le modèle de Baroni, Bayvel et Gibbens pour la protection $M:N$ qui nous sert de base pour la proposition d'un algorithme plus performant. Ce modèle mixte, car utilisant à la fois l'expression d'un flot en *arc-chemin* ou en *sommet-arc*, ne pouvant être résolu efficacement en entier, nous relâchons la contrainte d'intégrité des variables et nous proposons une méthode pour calculer une solution entière. Cette méthode, basée sur les travaux de Raghavan, nous permet d'obtenir une solution entière à une distance quantifiable par rapport à l'optimal avec une grande probabilité. Enfin, nous avons présenté les expérimentations qui comparent notre algorithme au modèle de Baroni, Bayvel et Gibbens.

CHAPITRE 5

Allocation de fréquence dans les réseaux satellitaires

Ce chapitre ne traite pas, contrairement aux précédents, d'allocation de ressources dans les réseaux optiques. Il s'agit ici de présenter un algorithme d'allocation de ressources pour les réseaux satellitaires[1]. La technologie employée n'a donc aucun rapport, mais les techniques mises en œuvre sont très proches. Dans les chapitres concernant la résolution des problèmes de routage, de groupage et de protection, un certain nombre d'algorithmes traitent des modèles utilisant la programmation linéaire, soit comme un moyen de formaliser le problème soit comme une méthode de résolution à part entière.

L'algorithme que nous proposons par la suite repose sur deux techniques principales : la génération de configurations admissibles pour les contraintes d'interférence par des heuristiques, la programmation linéaire mixte utilisant la génération de colonnes [**DW60**]. La solution que nous obtenons permet de prévoir un plan d'allocation admissible ayant des garanties d'optimalité. Elle permet aussi de mettre en évidence les configurations d'interférences qui entravent la génération de bonnes solutions.

Nous présentons dans un premier temps la problématique particulière de l'allocation de fréquence dans un réseau satellitaire, cette partie n'ayant volontairement pas été traitée dans le chapitre 1 de cette thèse. Nous proposons notre modèle, connexe aux autres modèles de la littérature. Nous détaillons ensuite les différentes étapes de l'algorithme et nous concluons en présentant l'intérêt des résultats obtenus.

5.1. Modélisation

L'allocation de fréquences dans un réseau de satellites consiste à prévoir un plan d'allocation en temps/fréquence pour un ensemble de terminaux ayant une configuration géométrique définie et soumis à des contraintes d'interférence. On cherche à minimiser la taille du plan de fréquences tout en garantissant que toutes les demandes des terminaux, en terme de bande passante et pour différents types, sont satisfaites.

5.1.1. Modèle

Nous considérons un système satellitaire géostationnaire multi-spot pour lequel un gestionnaire assigne les slots satellites de l'uplink MFTDMA (*Multi-Frequency Time-Division Multiple*

[1]Ce travail a fait l'objet du rapport de recherche [**AAG+04**] et des publications [**AAG+05a, AAG+05b**].

Access : accès multiple à répartition dans le temps commuté par satellite à modulation de fréquence) aux fournisseurs de service (opérateurs). Les fournisseurs de service eux-mêmes gèrent un parc de terminaux distribués sur la couverture satellitaire. Pour l'opérateur du canal radio, le satellite divise le temps et le spectre de fréquence. Géographiquement, les terminaux sont distribués sur des zones, elles-mêmes incluses dans des spots qui correspondent aux équipements de la réception du satellite. L'interférence radio impose des contraintes aux slots de transmission (abusivement notés slots dans la suite) qui peuvent simultanément être assignés dans différents spots qui utilisent la même fréquence. Un slot ne peut pas être assigné simultanément à plus d'une zone dans un même spot et peut être assigné à un opérateur dans une zone donnée seulement si l'interférence avec les autres zones actives est au-dessous d'un seuil donné. La planification des slots est statique mais peut être changée une fois par heure (due aux changements des demandes, d'une part, et des changements des conditions atmosphériques, d'autre part). Notre but est de maximiser la planification du système, c'est-à-dire la bonne utilisation des ressources disponibles, en terme de bande passante, pour satisfaire les demandes. Le problème peut se ramener à un problème de coloration : on peut donc conclure qu'il est $\mathcal{NP}$-complet [**Tof95, KC98**]. Nous proposons d'utiliser des heuristiques pour la génération de configurations admissibles ainsi qu'une approche utilisant la programmation linéaire mixte et la génération de colonnes.

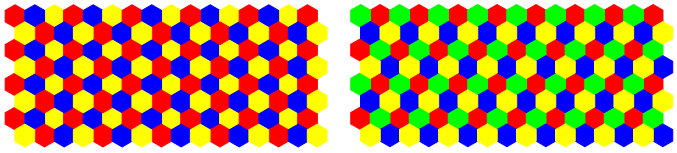
5.1.2. Problèmes connexes

Dans la grande majorité des cas, les références relatives à ce problème ont traité des modèles plus simples qui, dans certains cas, se résolvent à l'aide d'algorithmiques polynomiaux. Nous souhaitons cependant mentionner que des problèmes de nature semblable mais possédant une structure plus simple ont été également traités dans le contexte de l'établissement de programmes dans les réseaux ad hoc [**GK03**]. Concernant les autres aspects des satellites TDMA (*Time-Division Multiple Access*), le lecteur trouvera un état de l'art plus complet dans [**AGT02**] traitant de l'architecture, la synchronisation, ainsi que les problèmes relatifs à la couche physique ainsi qu'une discussion sur les techniques probabilistes d'évaluation des performances liées aux systèmes TDMA.

Notons que dans [**GW85**] on étudie le problème de la construction d'un plan d'allocation de fréquence TDMA. L'objectif est de permettre la transmission de ce plan en minimisant le nombre de fois où les bornes de transmission ont besoin d'être reconfigurées. Citons aussi un travail moins récent et traitant du même problème [**GCW82**]. Il concerne un système SS/TDMA (*Satellite Switched*) qui permet de multiplexer/démultiplexer une bande de fréquence en plusieurs sous-bandes. On souhaite aussi, dans ce cas, maximiser l'utilisation d'un tel système, en terme de bande passante. Les techniques utilisées, pour ces problèmes difficiles, sont principalement des heuristiques et des algorithmes de *branch-and-bound*.

5.1.3. Réutilisation spatiale

Le système que nous considérons tire avant tout parti d'une *réutilisation spatiale* de la bande passante dans la mesure où le taux d'interférence est acceptable. Cette réutilisation spatiale est possible de par la forme hexagonale des spots prévue par le modèle qui permet un arrangement géométrique régulier de ces spots. Ce modèle est très proche de celui de [**EE99**]. Ainsi, on divise la bande de fréquences en trois parties de tailles égales. On affecte alors, pour chaque spot, une couleur déterminant la bande de fréquence qu'il utilise comme représentée en figure 5.1 (a). De même, on peut réaliser la séparation de la bande de fréquence en quatre parties (figure 5.1 (b)). En fait, on affecte à chaque spot une couleur unique, qui fixe la bande



(a) 3 couleurs utilisées (b) 4 couleurs utilisées

FIG. 5.1 – Configuration spatiale des spots

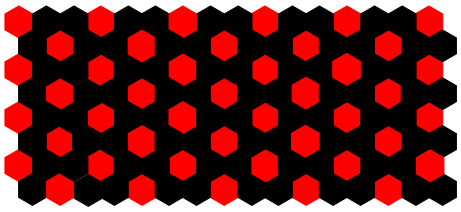


FIG. 5.2 – Configuration spatiale pour une unique couleur

de fréquence exclusive de couleurs dans laquelle il pourra communiquer. Cette affectation de couleurs est faite de sorte que deux spots (trop) voisins ne puissent émettre dans le même plan de fréquences, sans quoi l'interférence générée serait inacceptable.

Ainsi, au sein de chacune des différentes couleurs se définit un problème complètement indépendant de celui posé par les autres couleurs. Par conséquent, nous considérerons par la suite uniquement le problème d'une unique couleur sur trois (cas de la figure 5.1 (a)), c'est-à-dire sur les spots ayant cette même couleur et la bande de fréquence correspondante. On notera N ce nombre de spots, et B la taille de la bande passante. En pratique, on s'intéresse à $N \leq 32$. Cette configuration est représentée en figure 5.2.

La bande de fréquence est potentiellement réutilisée d'un spot à l'autre. Cette réutilisation n'est cependant possible que si le niveau d'interférence généré par l'ensemble des spots utilisant simultanément la ressource radio est acceptable.

5.1.4. Niveau d'interférence

Pour tenir compte des conditions réelles de la propagation radio, il est nécessaire de tenir compte de l'emplacement des terminaux au sein même d'un spot. Pour cela on introduit une notion de subdivision géographique d'un spot en *zones*. Chaque terminal est ainsi complètement caractérisé géographiquement par sa zone. En pratique, un spot a deux ou trois zones.

Chaque zone génère une certaine interférence $I(s, z)$ sur tous les autres spots (si s appartient au spot z alors $I(s, z) = 0$). Cette interférence est représentée sur la figure 5.3. Chaque zone a aussi un gain $G(z)$. Il faut absolument que sur tout canal utilisé, le gain divisé par la somme des interférences soit supérieur à un certain seuil σ, autrement dit, si $spot(z)$ est le spot contenant z, pour tout instant t et fréquence f, z peut être active en (t, f) seulement si

$$\frac{G(z)}{\sum_{z' \text{ active en } (t,f)} I(spot(z), z')} \geq \sigma \tag{5.1}$$

Dans les tests effectués par la suite, la valeur du gain en décibels est prise aléatoirement entre deux valeurs bornes. Pour l'interférence, on a distingué l'interférence de voisinage, générée sur les six spots les plus proches, et l'interférence globale, générée sur l'ensemble des

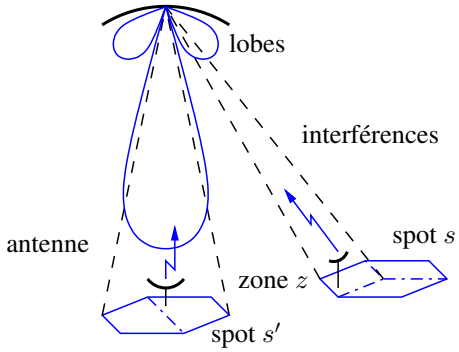


FIG. 5.3 – Interférences générées par un terminal de la zone z sur un terminal du spot s'.

Type	Nombre maximal de porteuses sur la bande	Nombre maximal de time-slots par trame
1	192	18
2	96	36
3	24	144
4	6	576

TAB. 5.1 – Valeurs test des types de terminaux.

spots, toutes deux étant prises aussi aléatoirement en dB dans leurs intervalles respectifs. La valeur finale de l'interférence est un barycentre à poids fixe des deux valeurs ci-dessus.

5.1.5. Types de terminaux et demande

Les terminaux ont des capacités variables d'émission. Ces capacités sont indiquées par la notion de type. Un type correspond à la largeur de la bande de fréquence qu'utilise un terminal. Le rapport entre les tailles de bande passante de deux types différents est un entier ou l'inverse d'un entier. L'intervalle de temps effectif d'un time-slot dépend du type utilisé, de sorte que le volume de données transmises lors d'un time-slot est une constante du problème. De manière équivalente, le produit largeur de bande d'une porteuse (la porteuse représente la bande de fréquence utilisée par le terminal) multiplié par la durée d'un time-slot est notée Δ et est une constante pour tous les types. Pour mémoire, les valeurs test du problème sont indiquées dans la table 5.1.

Chaque spot est divisé en zones, et chaque zone a une demande spécifique à satisfaire, exprimée en nombre de time-slots (ou unités de base de données) par types de terminaux.

$$\forall z_i, \forall k,\ d_{z_i}^{t_k} = \text{demande de } z_i \text{ en type } t_k$$

Un type de terminal t_k se caractérise par sa largeur de bande t_k^b et son temps de transmission t_k^t. On a, pour tout type k :

$$\Delta = t_k^b t_k^t$$

Notons enfin que l'on peut agréger sur une même zone toutes les demandes d'un même type. Ainsi, soit $d(z, t_k)$ la demande en time-slots de la zone z exprimée en nombre de time slots du type t_k.

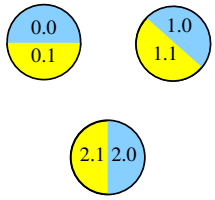


FIG. 5.4 – Petit exemple sur 3 spots.

Zone	Gain	I(.,Spot 1)	I(.,Spot 2)	I(.,Spot 3)
1.1	4	-	5	3
1.2	6	-	5	7
2.1	3	4	-	2
2.2	8	7	-	10
3.1	5	3	7	-
3.2	5	7	3	-

TAB. 5.2 – Tableau des gains et interférences des zones de l'exemple.

5.2. Exemple de problème

Pour introduire la problématique et les différents concepts définis précédemment, nous proposons l'étude de deux exemples qui montrent comment répondre à la demande en time-slots des différentes zones des spots.

5.2.1. Etude d'un cas simple

Pour bien comprendre la modélisation, on peut imaginer qu'il n'y a qu'un seul type de terminal. Le canal radio se découpe alors en "time-slots" - comme en GSM - où un certain nombre de zones peuvent émettre simultanément. La première question est de savoir : quelles zones peuvent alors émettre dans un time-slot et à une fréquence donnée ?

Prenons un exemple simple, illustré par la figure 5.4. Sur cette figure, chaque spot est découpé en deux zones qui ont une interférence spécifique sur les autres spots. Par exemple, on peut prendre le tableau numérique 5.2, qui précise, pour chaque zone, son gain ainsi que les interférences qu'elle génère sur les autres spots. Concentrons-nous maintenant sur la zone 1.1. Dans quelle mesure cette zone peut-elle émettre ? Cela dépend des états des zones qui interfèrent. Suivant que les zones 2.1, 2.2, 3.1 et 3.2 sont allumées ou pas, son rapport C/I va évoluer. La table 5.3 résume les 9 cas de figure qui peuvent se présenter.

En fixant le seuil d'admissibilité à 0.30, toutes les combinaisons possibles de spots avec deux zones allumées dans deux spots différents sont possibles, et trois combinaisons avec trois spots actifs sont possibles. Nous les montrons sur la figure 5.5. Le lecteur pourra vérifier que ces familles vérifient les conditions d'interférence pour chaque zone et que les autres configurations à trois spots actifs ne les vérifient pas.

Du point de vue de l'efficacité du système, il est clair que les configurations à trois zones allumées sont plus intéressantes. En effet, elles permettent de faire fonctionner simultanément trois spots, alors que les autres configurations ne permettent que deux fonctionnements en parallèle. Par la suite, nous appellerons les trois configurations de la figure 5.5 les familles 1, 2 et 3.

C/I (Zone 1.1)	Zone 2.1 allumée	Zone 2.2 allumée	Spot 2 éteint
Zone 3.1 allumée	0.57	0.40	1.33
Zone 3.2 allumée	0.36	0.29	0.57
Spot 3 éteint	1.00	0.57	-
C/I (Zone 1.2)	Zone 2.1 allumée	Zone 2.2 allumée	Spot 2 éteint
Zone 3.1 allumée	0.86	0.60	2.00
Zone 3.2 allumée	0.55	0.43	0.86
Spot 3 éteint	1.50	0.86	-
C/I (Zone 2.1)	Zone 1.1 allumée	Zone 1.2 allumée	Spot 1 éteint
Zone 3.1 allumée	0.25	0.25	0.43
Zone 3.2 allumée	0.38	0.38	1.00
Spot 3 éteint	0.60	0.60	-
C/I (Zone 2.2)	Zone 1.1 allumée	Zone 1.2 allumée	Spot 1 éteint
Zone 3.1 allumée	0.67	0.67	1.14
Zone 3.2 allumée	1.00	1.00	2.67
Spot 3 éteint	1.60	1.60	-
C/I (Zone 3.1)	Zone 1.1 allumée	Zone 1.2 allumée	Spot 1 éteint
Zone 2.1 allumée	1.00	0.56	2.50
Zone 2.2 allumée	0.38	0.29	0.50
Spot 2 éteint	1.67	0.71	-
C/I (Zone 3.2)	Zone 1.1 allumée	Zone 1.2 allumée	Spot 1 éteint
Zone 2.1 allumée	1.00	0.56	2.50
Zone 2.2 allumée	0.38	0.29	0.50
Spot 2 éteint	1.67	0.71	-

TAB. 5.3 – Détail des cas d'interférence possibles pour différentes zones.

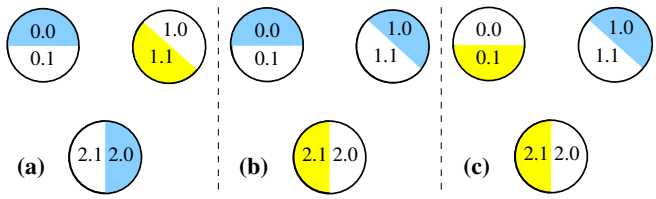


FIG. 5.5 – Ensemble des configurations d'émission possibles pour un seuil à 0.30.

Ainsi, si la demande est fixée à 100 time-slots par zone de manière uniforme, il est clair que chaque spot doit recevoir 200 time-slots et que, dans l'ensemble, il faudra au minimum 200 time-slots pour écouler cette demande. Or ce minimum peut être atteint. En effet, il suffit que dans les 100 premiers time-slots les zones 1.1, 2.2 et 3.1 émettent simultanément, puis dans les 100 suivants, les zones 1.2, 2.1 et 3.2.

5.2.2. Etude d'un cas plus complexe

Prenons maintenant la demande un peu plus complexe du tableau 5.4. On voit que dans cet exemple, chaque spot doit recevoir 200 time-slots. Mais il faut plus que 200 time-slots pour

Zone	Demande (time-slots)
1.1	50
1.2	150
2.1	50
2.2	150
3.1	150
3.2	50

TAB. 5.4 – Tableau des demandes de l'exemple.

Nombre de time-slots	Configuration (famille) utilisée
50	Zone 1.1 + Zone 2.2 + Zone 3.1
50	Zone 1.2 + Zone 2.1 + Zone 3.2
50	Zone 1.2 + Zone 2.2
50	Zone 1.2 + Zone 3.1
50	Zone 2.2 + Zone 3.1

TAB. 5.5 – Solution de l'exemple.

Nombre de time-slots	Configuration (famille) utilisée
100	Zone 1.1 + Zone 2.2 + Zone 3.1
50	Spot 1 + Zone 2.1 + Zone 3.2
50	Zone 1.2 + Zone 2.2
50	Zone 1.2 + Zone 3.1

TAB. 5.6 – Une description plus efficace de la solution de l'exemple.

écouler cette demande, car on ne peut pas utiliser de manière efficace les trois configurations citées plus haut en permanence. Une méthode néanmoins efficace peut consister à utiliser dans un premier temps 50 time-slots dédiés aux zones 1.1, 2.2 et 3.1, puis 50 aux zones 1.2, 2.1 et 3.2, puis 50 aux zones 1.2 et 2.2, puis 50 aux zones 1.2 et 3.1, puis 50 aux zones 2.2 et 3.1. En tout, c'est donc 250 time-slots qui seront utilisés pour écouler la demande. Cette solution peut se synthétiser comme expliqué sur le tableau 5.5, où on associe un nombre de time-slots à des configurations. Dans la suite du document, on associera le terme "famille" à ces configurations admissibles, et on déterminera le nombre de time-slots où ces familles seront utilisées.

Pour une meilleure résolution du problème, d'autres notions pourront être utiles. Par exemple, notons que les familles 2 et 3 diffèrent seulement par le spot 1. En fait, il est correct de dire qu'il est possible d'allumer simultanément *tout* le spot 1 avec les zones 2.1 et 3.2. Ainsi, nous allons créer, dans des conditions mathématiques qui seront précisées dans la suite, des configurations qui en résument plusieurs autres en laissant de la flexibilité sur la zone que l'on allume au sein d'un spot. Il est également possible, pour décrire une solution du problème, d'utiliser une famille qui a "trop" de zones allumées, dans le sens où certaines resteront inutilisées pendant l'utilisation effective du système. Ainsi, on pourrait décrire la solution du problème de manière plus efficace comme le montre le tableau 5.6.

5.3. Résolution du problème d'interférence

La section qui suit introduit plus formellement la notion de famille. Ainsi, nous pouvons définir les familles qui sont dites valides (et inversement non-valides) vis-à-vis du critère d'interférence.

5.3.1. Calcul du niveau d'interférence

La solution au problème des questions d'interférence vient de la constatation simple que pour toute ressource radio, et donc pour tout couple temps-fréquence (t, f), il existe une famille Z de zones simultanément actives, qui vérifient donc :

$$\forall z \in Z \quad \frac{G(z)}{\sum_{z' \in Z} I(spot(z), z')} \geq \sigma.$$

Il est alors assez naturel d'utiliser ce concept d'émissions non concurrentes de manière similaire à ce qui se fait pour le coloriage de graphes [**GLS81**]. Dans le cas du coloriage de graphe, on s'appuie sur un ensemble de familles de sommets indépendants. Ici, on utilise des familles de zones pouvant communiquer simultanément.

Le nombre de familles vérifiant ce critère est en pratique très grand pour que l'on puisse les décrire de manière exhaustive. Par ailleurs, il sera difficile de bâtir une solution uniquement en se basant sur ce concept élémentaire de famille. Par conséquent nous avons introduit la notion de super-zone au sein d'un spot. Une super-zone contient plusieurs zones d'un même spot. Si une famille contient une super-zone, cela indique que chacune des zones qu'elle contient peut émettre avec les autres zones en respectant les conditions d'interférence. Les paramètres d'une super-zone ζ regroupant deux zones z et z' d'un même spot sont les suivants :

$$\begin{array}{lcll} G(\zeta) & = & \min(G(z), G(z')) & \\ I(s, \zeta) & = & \max(I(s, z), I(s, z')) & \text{pour tout spot } s \end{array}$$

La méthode de résolution consiste donc, dans un premier temps, à générer des familles F_i constituées de zones ou super-zones de spots distincts.

5.3.2. Heuristique de génération de familles valides

Pour un spot contenant 3 zones, nous définissons le statut qui peut prendre l'une des valeurs suivantes :

0 : le spot est éteint ;
1 : la zone 0 du spot vérifie le critère d'allocation et est donc allumable ;
2 : la zone 1 du spot vérifie le critère d'allocation et est donc allumable ;
3 : les zones 0 et 1 vérifient toutes les deux le critère d'allocation et sont donc allumables ;
4 : la zone 2 du spot vérifie le critère d'allocation et est donc allumable ;
5 : les zones 0 et 2 vérifient toutes les deux le critère d'allocation et sont donc allumables ;
6 : les zones 1 et 2 vérifient toutes les deux le critère d'allocation et sont donc allumables ;
7 : les zones 0, 1 et 2 vérifient toutes les trois le critère d'allocation et sont donc allumables.

L'introduction du statut d'un spot donne plus de liberté et de souplesse lors de l'allocation de ressources. Ainsi, si un spot ayant 3 zones a un statut 7, ceci veut dire que nous pouvons allouer les ressources à n'importe laquelle des 3 zones.

Pratiquement, une famille est représentée par un vecteur, dont la taille est égale au nombre de spots, et dont les éléments sont les statuts des spots. Nous désignons alors une famille x/y, une famille telle que, sur y spot consécutifs, x sont allumées et $y - x$ sont éteints. Comme nous

voulons maximiser le nombre de zones allumées, nous commençons par générer les 7 familles 6/7 dans lesquelles tout spot allumé s a le statut $2^{nbZones(s)} - 1$, les spots éteints ayant le statut 0. Nous testons successivement la validité de ces familles et les séparons en deux tas : d'un côté les familles 6/7 valides et de l'autre côté les familles 6/7 non valides (en terme de niveau d'interférence de la section 5.3.1). Nous pouvons faire de même avec les familles 5/7, 4/7, etc., selon le seuil d'interférence σ considéré. Pour qu'une famille non valide devienne valide, il faudrait que certaines de ses zones allumées soient éteintes. Nous proposons l'heuristique suivante pour rendre une famille valide :

(1) choisir au hasard une famille du tas de familles non valides ;

(2) tant que la famille n'est pas valide :
   (a) choisir au hasard un spot,
   (b) si le statut est non nul et que le spot est non valide, éteindre au hasard une des zones allumées et sauvegarder le numéro du spot ainsi modifié ;

(3) essayer pour un certain nombre de fois de rallumer des zones qui ont été éteintes en 2. et tester la validité de la famille après chaque essai : une modification n'est adoptée que si la famille est valide ;

(4) comparer la famille valide obtenue en 3. à celles qui se trouvent dans le tas des familles valides. En cas de redondance, incrémenter un compteur de redondances et rejeter la famille, sinon ajouter la famille au tas des familles valides. Revenir à 1. pour générer une autre famille.

Cet algorithme s'arrête soit quand le nombre voulu de familles valides a été atteint, soit quand le compteur de redondances atteint une certaine valeur maximale.

5.4. Résolution des contraintes de placement des capacités sur le canal radio

Les demandes de chaque zone sont de différents types. Ces types ont une largeur de bande et une durée différente, comme défini en section 5.1.5. Nous proposons dans cette section un algorithme de placement qui permet de répartir ces différents types de demandes sur la bande passante du satellite.

5.4.1. Preuve de la faisabilité du placement

Les contraintes au niveau du canal radio mettent en jeu deux grandeurs : la largeur de bande B et le temps. Lorsqu'on planifie l'utilisation d'un terminal de type t_k pendant un time-slot, on utilise schématiquement un rectangle d'une certaine aire dans l'intervalle temps-fréquence.

Ainsi, si on identifie les types par des indices de 1 à τ ordonnés par largeur de bande décroissante, et si on note x^{t_i} le nombre de time-slots utilisés dans le spot en question par le type i, on a :

$$\sum_{i \in \{1 \ldots \tau\}} x^{t_i} \leq \frac{BT}{\Delta}.$$

Le résultat suivant nous sert à établir les propriétés d'un remplissage de time-slots :

Lemme 4. *Soit $G = (V, E)$ le graphe orienté avec $V = \{1, \ldots, \tau\}$ et $E = \{(j, k) : j < k\}$ avec $w_{(j,k)} = \frac{t_j^b}{t_k^b}$. Alors tout chemin dans G de 1 à τ a un poids inférieur à $w_{(1,\tau)}$.*

Algorithme 10 Algorithme de placement

Entrée: Une pile ordonnée H de types $\{t_1, \ldots, t_1, t_2, \ldots, t_2, \ldots, t_\tau, \ldots, t_\tau\}$. Un ensemble ordonné R de rectangles $(1, \tau)$ ordonnés de la gauche vers la droite et du haut vers le bas
Sortie: Un placement sur $B \times T$

```
Soit r le premier rectangle (1, τ) dans R
tant que T n'est pas vide faire
    Dépiler t_i de H
    si t_i ne peut pas être placé dans r alors
        Sélectionner le rectangle suivant r de R
    fin si
    Mettre t_i dans r le plus à gauche possible, puis le plus haut possible
    si t_i et t_j = head(H) sont des types différents alors
        Sauter à la multiplicité w_(i,j) en remplissant avec des types vides t_i {gaspillage}
    fin si
fin tant que
```

Démonstration. On note que le graphe G est transitif. Par ailleurs, pour tous réels x et y tels que $x \geq 1$ et $y \geq 1$, on a $x-1+y-1 = xy-1-(x-1)(y-1) \leq xy-1$. Ainsi, si $(i,j) \in E$ et $(j,k) \in E$, alors $(i,k) \in E$ et $w_{(i,k)} \geq w_{(i,j)} + w_{(j,k)}$. D'où le résultat par transitivité. □

Par la suite, on montre qu'un chemin dans ce graphe correspond en fait à des pertes à cause de la structure géométrique du problème.

Proposition 5. *Il est possible de placer, sur l'espace temps fréquence, x^{t_i} time-slots de type i, pour $i \in 1, \ldots, \tau$ si*

$$\sum_{i \in \{1,\ldots,\tau\}} x^{t_i} \leq \frac{BT}{\Delta} - \frac{t_1^b}{t_\tau^b} + 1.$$

Démonstration. Nous donnons le squelette de la preuve. L'espace de la trame peut être découpé en "rectangles $(1, \tau)$" donc le côté "fréquence" est la largeur de bande maximale et le côté "temps" est le temps de time-slot maximal. On remplit les rectangles $(1, \tau)$ par ordre croissant de type.

Si un rectangle $(1, \tau)$ contient des types i et des types j on l'autorise à "perdre" l'espace $w_{(i,j)}$: en effet il se subdivise lui-même en rectangles (i, j) qui vont être de types croissants de i à j. Au total, au maximum on perd la somme des poids dans un chemin du graphe du lemme précédent. □

Par la suite on posera $\delta = \frac{t_1^b}{t_\tau^t} - 1$. On note que pour les données test du tableau 5.1, cette contrainte permet de résoudre le problème du placement en sacrifiant moins de 1% de la bande passante. Il est possible d'être encore plus performant en essayant le calcul ci-dessous avec une valeur de δ inférieure en faisant le pari raisonnable que l'arrangement devrait pouvoir se faire.

5.4.2. Algorithme de placement

L'algorithme 10 donne les idées principales de la politique de placement utilisée. Le placement est réalisé en remplissant la suite de rectangles $(1, \tau)$ de gauche à droite et de haut en bas et de même, à l'intérieur d'un rectangle, on remplit encore de gauche à droite et de haut en bas. On ne considère dans cet algorithme que le cas des familles mono-typées. c'est-à-dire ne produisant qu'un seul et même type. Ainsi, on ne considère que les types t_1 à t_τ.

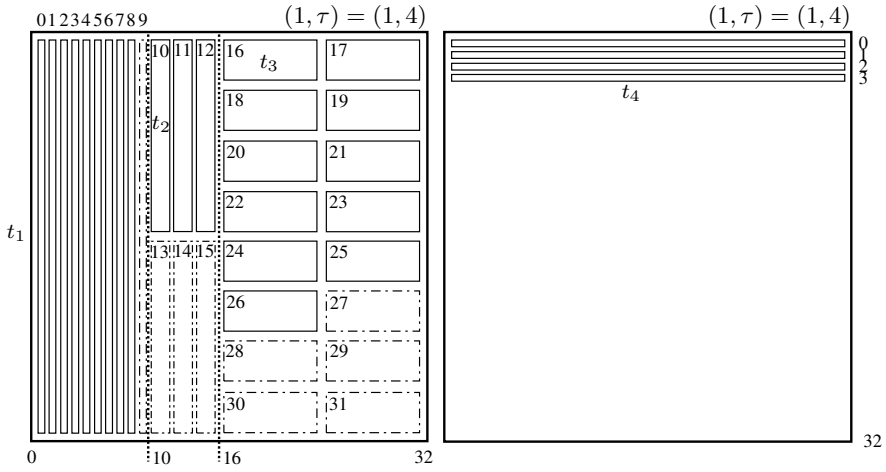


FIG. 5.6 – Étapes de l'algorithme de placement

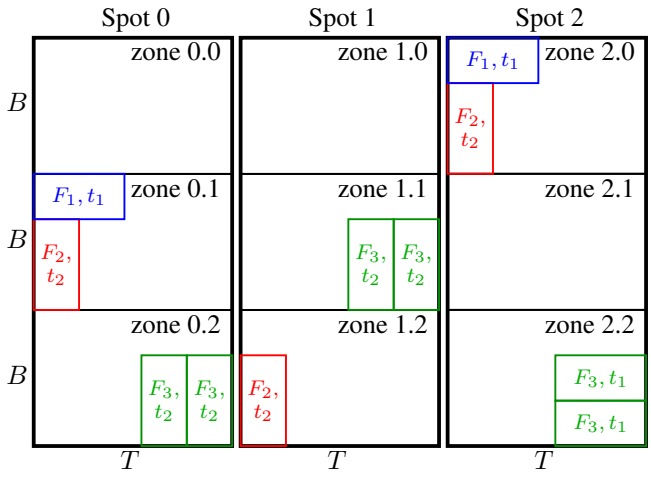


FIG. 5.7 – Un exemple d'arrangement global des familles

Les ordres de multiplicité de l'exemple de la figure 5.6 sont 8 entre les types 1 et 2, et 2 entre les types 2 et 3, i.e. $t_1^b = 8t_2^b = 16t_3^b$ et $t_3^t = 2t_2^t = 16t_1^t$. Les rectangles dessinés en pointillés sont des "espaces perdus" sur l'espace temps-fréquence alors que les rectangles continus sont des slots placés, et typés. L'exemple de la figure 5.6 étant petit et la configuration étant choisie volontairement mauvaise, l'espace perdu est très important : pour placer 25 time-slots, on perd 9 time-slots. Les 15 time-slots disponibles à la fin du placement sont inutilisés, mais peuvent l'être pour d'autres nouvelles demandes.

La figure 5.7 montre un placement possible des ressources radio. Si $F_i^T(s) = t_k$, on notera (F_i, t_k) dans le rectangle concerné de la figure et c'est pour cela qu'on retrouve cette notation dans toutes les zones actives de la famille (ici, les zones des spots 0,1 et 2 pour la famille F_2 et le spot 0 et 2 pour la famille F_1). La contrainte de capacité de chaque zone, en terme de bande passante et de temps est assurée par la contrainte de surface des rectangles (F_i, t_k) sur le rectangle $B \times T$.

Une famille peut avoir plusieurs types différents de terminaux, selon les spots considérés. Dans le cas de la figure 5.7 par exemple, pour les rectangles (F_3, t_1) et (F_3, t_2) on dit que la famille F_3 est 2-typée. Ces familles ont un ordre de multiplicité spécifique. Si t_k est le type de la famille ayant la largeur de bande maximum et $t_{k'}$ le type ayant la largeur de bande minimum, alors l'ordre de multiplicité est de $F_i^M = \frac{t_k^b}{t_{k'}^b} \in \mathbb{N}^* - \{1\}$.

5.5. Satisfaction de la demande

L'algorithme de placement permet d'agencer les différents types de demandes allouées pour le satellite, ce qui résout une partie du problème. Il faut maintenant déterminer quelles familles sont utilisées pour émettre sur la bande de fréquence et pour chaque famille, quelle type de demande elle utilise. Pour résoudre ce problème, nous introduisons un programme linéaire dont l'objectif est de satisfaire les zones en demande de chaque type.

5.5.1. Programme linéaire $\mathcal{P}$

Dans cette section nous présentons le programme linéaire qui permet de calculer une solution basée sur des familles typées. Sans perte de généralité, on considérera qu'un spot possède trois zones, pour décrire ce programme linéaire. On modélise les contraintes de satisfaction de la demande à l'aide des équations (5.3)-(5.5). L'équation (5.2) fournit la contrainte d'espace temps fréquence qui permet de s'assurer que l'ensemble des time-slots rentrent dans l'espace temps-fréquence, condition suffisante d'après le théorème 5.

Les variables du programme linéaire, dénoté $\mathcal{P}$, sont les x_{F_i}, qui représentent le nombre de fois où des familles typées sont utilisées. Ce sont des variables qui doivent être entières. Soit $\mathcal{I}$ l'ensemble courant des familles typées disponibles pour résoudre $\mathcal{P}$. Nous notons $d(z, t_k)$ la demande en type t_k, comme défini dans la section 5.1.5. Soit $F_i^A(z) = \textit{on}$ la notation exprimant qu'une zone z peut être active et à l'inverse, $F_i^A(z) = \textit{off}$ lorsqu'elle ne peut l'être. $\mathcal{P}$ est alors défini comme le problème de minimisation, $\min\ J$, qui suit :

Programme linéaire 14.

$$J = \sum_{i \in \mathcal{I}} F_i^M x_{F_i} \leq \frac{T \times B}{\Delta} - \delta \tag{5.2}$$

$$\forall j \in [1, \tau],\ \forall z \in s,\ \sum_{i \in \Gamma(z,j)} F_i^M x_{F_i} \geq d(z, t_j) \tag{5.3}$$

$$\forall j \in [1, \tau],\ \forall z, z' \in s,\ \sum_{i \in \Gamma(z,z',j)} F_i^M x_{F_i} \geq d(z, t_j) + d(z', t_j) \tag{5.4}$$

$$\forall j, \in [1, \tau],\ \forall\ s,\ \sum_{i \in \Gamma(z,z',z'',j)} F_i^M x_{F_i} \geq d(z, t_j) + d(z', t_j) + d(z'', t_j) \tag{5.5}$$

avec :

$$\begin{aligned}
\Gamma(z, j) &= \left\{ i \in \mathcal{I} / F_i^T(s) = t_j,\ F_i^A(z) = \text{on} \right\} \\
\Gamma(z, z', j) &= \left\{ i \in \mathcal{I} / F_i^T(s) = t_j,\ (F_i^A(z) = \text{on}\ \ \textit{or}\ \ F_i^A(z') = \text{on}) \right\} \\
\Gamma(z, z', z'', j) &= \left\{ i \in \mathcal{I} / F_i^T(s) = t_j,\ \exists z \in s / F_i^A(z) = \text{on} \right\}
\end{aligned}$$

Si la contrainte d'aire (5.2) n'est pas satisfaite, aucune solution entière ne peut être trouvée. C'est pourquoi nous choisissons la surface d'occupation comme la fonction objectif que nous

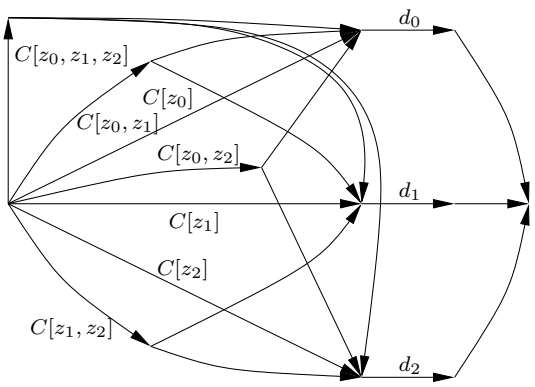


FIG. 5.8 – Modélisation des contraintes de zones par un flot.

minimisons. Ainsi, en minimisant J, on maximise la réutilisation des ressources du canal radio et donc la bande passante totale du système.

Proposition 6. *Les équations* (5.3)-(5.5) *garantissent la satisfaction des demandes en type* t_j.

Démonstration. La satisfaction des demandes en types t_j peut se calculer sur un flot depuis une source s, en passant par trois arcs (ou $nbZones$, dans le cas où il y a $nbZones$ zones) de capacités respectives $d_0 = d(t_j, z_0)$, $d_1 = d(t_j, z_1)$, et $d_2 = d(t_j, z_2)$, comme indiqué en figure 5.8. Les capacités des autres arcs notées par $C[z_0, z_1, z_2]$, $C[z_k, z_{k'}]$ pour $k \neq k'$, $\{k, k'\} \subset \{0, 1, 2\}$ et $C[z_k]$, $\{k\} \subset \{0, 1, 2\}$, sont données par :

$$\begin{aligned} C[z_0, z_1, z_2] &= \sum_i x_{F_i}^{t_j} \times U[Fi, \{z_0, z_1, z_2\}] \\ C[z_k, z_{k'}] &= \sum_i x_{F_i}^{t_j} \times U[Fi, \{z_k, z_{k'}\}] \\ C[z_k] &= \sum_i x_{F_i}^{t_j} \times U[Fi, \{z_k\}] \end{aligned}$$

où $U[Fi, Z]$ est égal à 1 si F_i peut activer l'une de ses zones de l'ensemble Z, et 0 sinon.

Les capacités de tous les autres arcs de la figure sont supposées infinies. En effet, par le théorème de Ford Fulkerson [**FF62**], il existe un flot maximum et entier allant de la source au puits, qui est égal à la cardinalité d'une coupe minimale. Or les coupes de tailles finies sont au nombre de 8 (ou 2^w dans le cas de w zones), suivant le choix des arcs de capacité d_1, d_2 et d_3, et vérifier ces équations permet de s'assurer que le flot est bien écoulé. L'une de ces équations est triviale puisqu'elle stipule que le flot des zones doit être inférieur à $d_1 + d_2 + d_3$. Les sept autres sont vérifiées par notre programme linéaire. □

5.5.2. Assignation optimales des types aux familles

Il existe τ^N différentes manières d'assigner des types à des familles non typées F_i. On ne peut donc inclure toutes ces familles dans $\mathcal{P}$ et nous proposons d'utiliser le concept très connu de génération de colonne [**DW60**]. Une colonne correspond alors à une famille typée et valide. La solution fractionnaire optimale est obtenue lorsque $\mathcal{I}$ contient toutes les familles possibles :

il s'agit donc d'initialiser $\mathcal{I}$ à un ensemble de familles typées selon la méthode décrite en section 5.3.2, puis d'utiliser les propriétés du dual pour identifier des nouvelles colonnes qui sont ajoutées à $\mathcal{I}$.

Réécrivons $\mathcal{P}$ sous la forme suivante :

$$\begin{array}{ll} \text{Minimiser} & f = c \cdot x \\ \text{Tel que} & \left\{ \begin{array}{lcl} Ax & = & b \\ x & \geq & 0 \end{array} \right. \end{array}$$

Soit A_B la matrice extraite du système d'équation et qui forme les colonnes de la base et soit x_B le vecteur des familles associées. De même, A_N et x_N sont les matrices et vecteurs de la partie hors base, c_B et c_N les coefficients de la fonction objectif. On peut alors écrire :

$$A_B x_B + A_N x_N = b \quad \text{et} \quad f = c_B x_B + c_N x_N.$$

Alors :

$$\begin{array}{rcl} x_B & = & A_B^{-1} b - A_B^{-1} A_N x_N, \\ f & = & c_B A_B^{-1} b + (c_N - c_B A_B^{-1} A_N) x_N. \end{array}$$

Ces équations donnent une solution au système de départ, si $x_N = 0$. Le système est alors optimal si et seulement si :

$$c_N - c_B A_B^{-1} A_N \geq 0.$$

Dans ce cas, le système peut être amélioré si l'on trouve un coefficient négatif dans le vecteur précédent. Dans notre cas, ce vecteur est particulier. On a notamment, $c_{\alpha_j} = F^M_{\alpha_j}$. Trouver une telle famille consiste à trouver une constante $K_{\mathcal{F}}$ et une fonction $\kappa_{\mathcal{F}}(F_i^T(s), s), s \rightarrow \mathbb{R}$ qui améliore un des coefficients du vecteur si l'on pose :

$$\begin{array}{rcl} c_i - c_B A_B^{-1} A_i & = & F_i^M \left(c_i / F_i^M - \sum_{s \text{ spot}} c_B A_B^{-1} P_s A_i / F_i^M \right). \\ & = & F_i^M \left(K_{\mathcal{F}} - \sum_{s \text{ spot}} \kappa_{\mathcal{F}}(F_i^T(s), s) \right). \end{array}$$

Dans les équations précédentes, P_s est la projection sur l'espace vectoriel de la base. On obtient alors le résultat suivant :

Proposition 7. *Le programme $\mathcal{P}$ avec l'ensemble restreint de familles de $\mathcal{I}$ est améliorable avec des nouvelles familles valides si et seulement si il existe une famille non typée $\mathcal{F}$ telle que :*

$$K_{\mathcal{F}} - \sum_{s \text{ spot}} \max_{t \text{ type}} \kappa_{\mathcal{F}}(t, s) < 0. \tag{5.6}$$

Ce résultat est particulièrement intéressant car il permet d'identifier des familles non-typées. On peut ensuite utiliser les coefficients du dual pour identifier les types les plus intéressants pour chaque zone, c'est-à-dire pour chaque ligne du vecteur, et créer la meilleure famille typée possible.

5.6. Structuration des étapes de l'algorithme

Cette partie résume les différentes étapes de la totalité de l'algorithme. Elle exploite les différents résultats présentés ci-avant et présente leur agencement algorithmique. Nous utilisons

un découpage de l'algorithme global en deux sous-programme maître/esclave, permettant de réaliser la génération de colonnes.

5.6.1. Le programme esclave

À partir d'un ensemble de familles non typées $\mathcal{I}$, le programme esclave assigne des types et retourne la solution exacte de $\mathcal{P}$ parmi l'ensemble de tous les typages possible. Au départ, les familles sont mono typées ou 1-typées. La solution trouvée permet de construire, à partir du dual et comme décrit en section 5.5.2, un 2-typage améliorant encore la solution. Le programme linéaire est résolu de nouveau et des itérations ont encore lieu s'il est possible de générer un 2-typage améliorant. On continue ce processus jusqu'au τ-typage. Lorsque l'optimum est obtenu, le programme esclave se termine.

5.6.2. Le programme maître

Le programme maître exploite les propriétés du dual, comme expliqué en section 5.5.2 pour générer des familles non-typées améliorantes. Ainsi, on améliore l'ensemble $\mathcal{I}$ qui pourra ensuite être passé au programme esclave. Pour déterminer la meilleure famille améliorante, il faut en général résoudre un problème plus ou moins difficile. Dans notre cas, on peut "presque" se contenter d'énumérer toutes les possibilités, comme expliqué ci-après.

Chaque spot peut au plus être dans un état parmi quatre, s'il est éteint ou que l'une de ses trois zones est allumée. Ainsi cela nous donne, pour huit spots, un total de $4^8 = 65536$ configurations à tester, ce qui est très raisonnable. Ce nombre augmente rapidement avec le nombre de spots potentiellement actifs. Pour pallier ce problème, et laisser une chance au programme de trouver un résultat optimal en temps rapide même en présence d'un nombre plus important de spots, un test élagué permettant de trouver les familles pouvant améliorer la solution plus facilement. Plusieurs étapes sont alors mises en œuvre, en particulier :

- Un "élagueur" sélectionne les zones au sein d'un spot. En particulier, si plusieurs zones se voient affecter le même gain, une parmi celles-ci est préselectionnée pour une recherche exhaustive ultérieure.
- Une fois les p familles de meilleur gain sélectionnées, celles-ci sont "réaugmentées" s'il est possible de rendre allumables plusieurs zones au sein d'un spot tout en respectant les contraintes de validité de la famille.

La figure 5.9 montre le processus de recherche élaguée.

5.6.3. Solution entière de $\mathcal{P}$

La résolution du programme esclave permet de générer les colonnes qui donne le meilleur résultat fractionnaire, pour chaque cas de k-typage. Toutes ces colonnes sont réintroduites dans un nouveau programme linéaire entier et sont candidates pour la résolution du problème sous sa forme entière. Nous savons qu'il existe une solution avec un nombre de variable non nul x_{F_i} au moins égal au nombre de lignes du programme linéaire [**Chv83**] (Théorème 9.3, page 145). Par exemple, dans le cas de 8 spots, seulement 224 variables sont utilisées (896 pour 32 spots) ; ainsi, un simple arrondi des variables donnera une solution entière qui ne s'éloignera pas plus de 2.1% de la solution fractionnaire (8.3% pour le cas 32 spots).

En pratique, la résolution du programme linéaire se fait en utilisant la MIP de CPLEX SOLVER™. Pour obtenir des résultats en un temps raisonnable, nous fixons la tolérance d'éloignement de l'optimum à 1% pour toutes nos expérimentations.

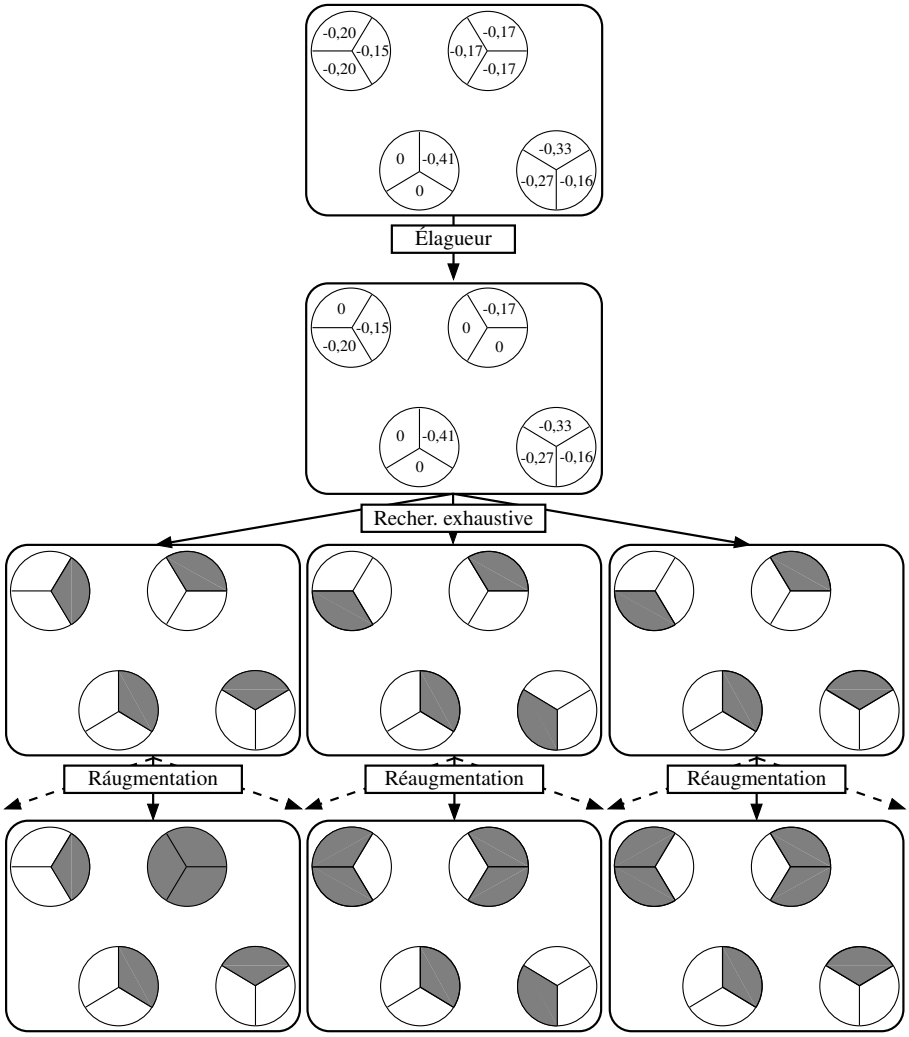


FIG. 5.9 – Recherche élaguée de familles améliorantes.

5.6.4. Résumé de l'algorithme d'allocation

Cette partie résume le déroulement général de l'algorithme. La figure 5.10 représente par un rectangle (respectivement un ovale) les parties correspondant à une tâche (respectivement une action ou une décision). L'algorithme débute dans le rectangle le plus à gauche par la génération de familles valides mais non typées comme expliqué en section 5.3.2. Ces familles sont données directement au programme esclave. L'esclave opère comme décrit en section 5.6.1 : les familles sont typées, le programme linéaire $\mathcal{P}$ est résolu et l'esclave itère jusqu'à obtenir l'optimalité. Les familles utilisées dans la solution sont stockées pour la résolution du programme linéaire final. Ensuite, le programme maître vérifie l'optimalité globale à l'aide du critère de l'équation (5.6). Si l'optimalité n'est pas atteinte, le programme maître utilise l'élagueur décrit en section 5.6.2 pour générer des nouvelles familles. La combinaison maître-esclave continue à optimiser la solution jusqu'à l'optimum. Ensuite, un programme linéaire final est construit comme décrit en section 5.6.3 et est résolu en entier. Enfin, on termine l'algorithme en plaçant les time-slots des familles typées obtenues en utilisant l'algorithme 10 de la section 5.4.2.

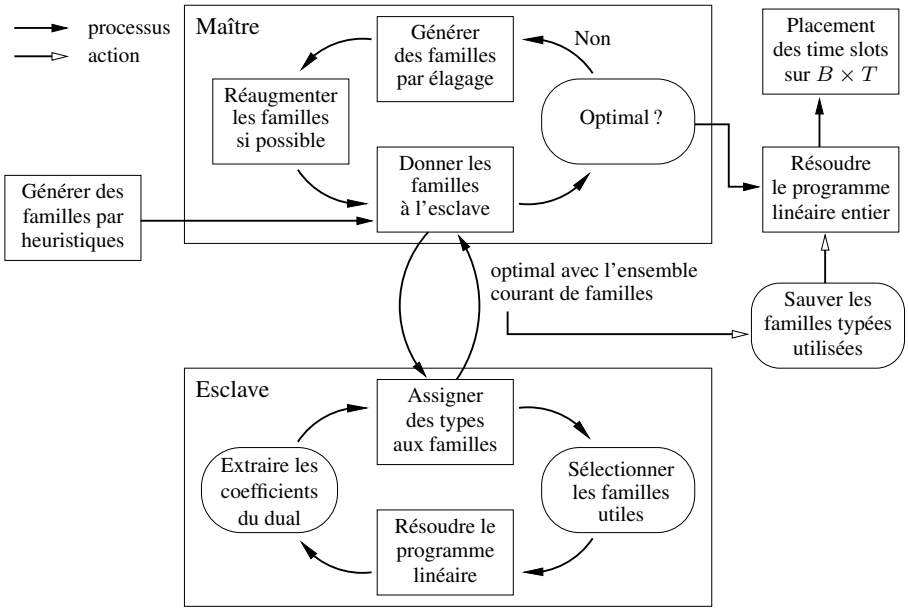


FIG. 5.10 – Schéma récapitulatif de l'algorithme d'allocation

5.7. Résultats expérimentaux

Nous présentons dans cette section les résultats obtenus avec la méthode résumée en section 5.6.4. Nous avons testé différents types de configurations entre 8 et 32 spots. La demande a été générée suivant les exemples précédemment fournis par ALCATEL SPACE INDUSTRIES, et les Interférences ainsi que les gains, ont été tirés suivant une loi uniforme en dB, selon les spécifications données par ALCATEL SPACE INDUSTRIES. L'interférence a été choisie, pour 85% de sa valeur, comme une interférence de voisinage (les six spots voisins) et pour 15%, comme une interférence globale ($\gamma = 0.85$).

La sortie de notre programme permet de dessiner, sur l'espace temps-fréquence, le plan d'allocation des time-slots alloués, de la même manière que le schéma de la figure 5.6. Le résultat présenté en figure 5.11 a été calculé à partir de données réelles fournies par ALCATEL SPACE INDUSTRIES.

5.7.1. Résultats avec 8 spots

Dans le cas de 8 spots, avec des temps de calcul de l'ordre de la minute sur une machine du type PENTIUM™ IV, nos programmes peuvent trouver la solution fractionnaire optimale. Ce résultat est particulièrement satisfaisant, car il permet d'analyser avec précision les causes d'interférence au niveau des spots. Sur la figure 5.12 nous montrons les valeurs d'aires minimales trouvées en fonction du seuil. La figure met clairement en évidence le fait qu'à certaines valeurs pivot, la valeur d'aire consommée augmente brutalement. Cela est dû au fait qu'à partir de certaines valeurs de seuil, des familles ne sont plus valides et leur absence dégénère la solution. Le tableau 5.7 détaille leurs structures.

Ce résultat a évidemment un impact très fort sur la conception des antennes, puisque l'on est capable de mettre en évidence les configurations d'interférences qui entravent la génération de bonnes solutions.

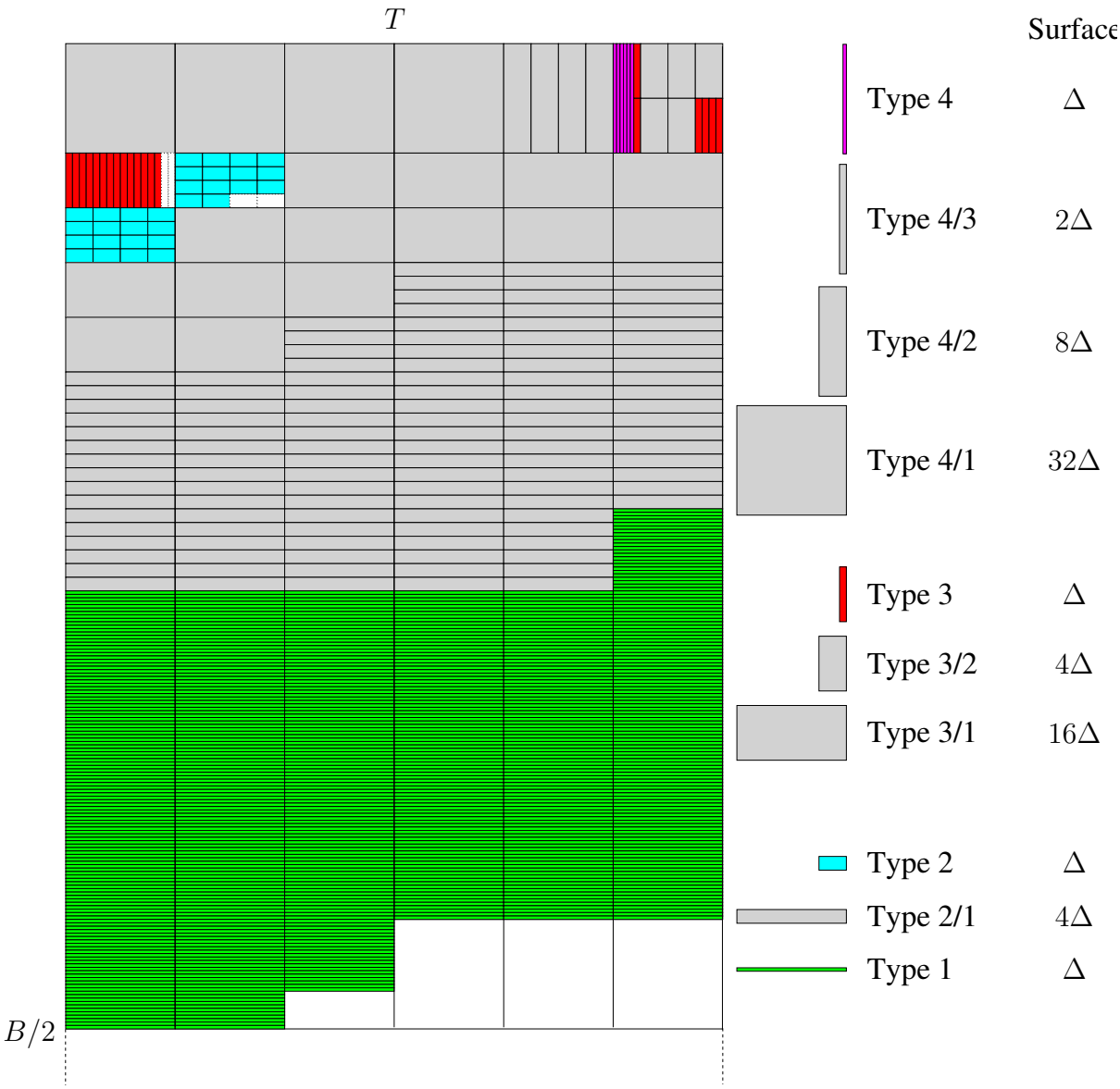


FIG. 5.11 – Un exemple de résultat d'allocation de ressources.

5.7.2. Résultats avec 32 spots

Pour une configuration de 32 spots, nous préconisons un fonctionnement non-optimal partant d'un nombre de familles limité. Des tests ont montré que la procédure de recherche d'une solution fractionnaire optimale pouvait grandement améliorer la solution. Néanmoins, le nombre de configurations possibles rendent alors les résultats peu présentables. C'est pourquoi nous avons décidé de simplement mettre en évidence des résultats avec des familles valides générées aléatoirement. Leur nombre tend à augmenter le temps de résolution tout en améliorant la solution. Il est donc possible de trouver un compromis pour l'élaboration d'une solution en temps réel.

Les résultats présentés sur la figure 5.13 présentent la valeur de l'aire obtenue, c'est-à-dire le nombre de time-slots total en fonction du nombre de familles générées aléatoirement. Plus

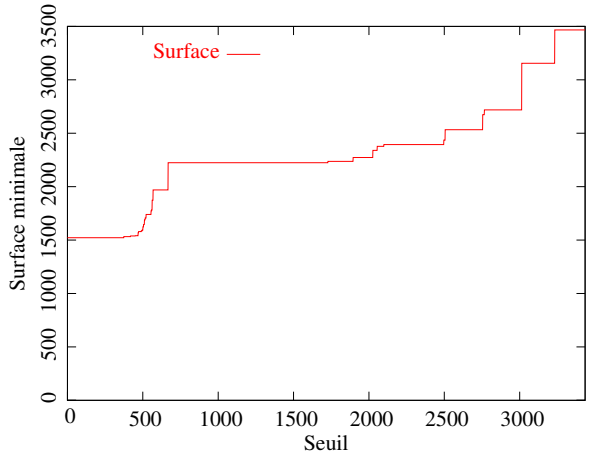


FIG. 5.12 – Valeurs d'aire obtenues sur l'ensemble des seuils possibles.

Seuil	Spot 0			Spot 1			Spot 2			Spot 3			Spot 4			Spot 5			Spot 6			Spot 7		
	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
373	X					X						X				X								
373	X					X						X						X						
418	X					X						X												
423	X					X										X								
450		X		X							X					X								
450		X		X								X					X							
450		X		X							X										X			
469			X	X								X					X							
472			X	X							X					X								
472			X	X							X										X			
472				X							X					X					X			
472					X						X					X					X			
490		X		X							X													
496				X								X				X					X			
501			X	X								X				X								
505			X	X												X					X			
510				X				X										X						

TAB. 5.7 – Familles perdues en fonction des seuils (X=zone éteinte).

ce nombre est important, plus l'aire est réduite. Les différents niveaux de seuil correspondent à σ, le seuil d'interférence défini en section 5.1.4. Plus σ est grand, plus l'interférence générée est forte et nécessite des familles ou de moins en moins de zones sont allumées. Le nombre de time-slots nécessaires à la demande augmente donc d'autant plus. Enfin, en figure 5.14, nous présentons les temps d'exécution de notre algorithme qui croît linéairement avec le nombre de familles générées.

Conclusion

Ce chapitre a présenté un problème d'allocation de fréquences dans un réseau satellitaire MFTDMA. Nous avons présenté le modèle complet du problème de la détermination d'un plan de fréquences permettant d'identifier les différents sous problèmes, à savoir, la résolution du problème d'interférences, puis l'allocation de time-slots pour les terminaux communiquant avec le satellite. Nous montrons alors comment utiliser la notion de famille typée pour modéliser ces deux problèmes conjointement dans un même programme linéaire. À partir de ce

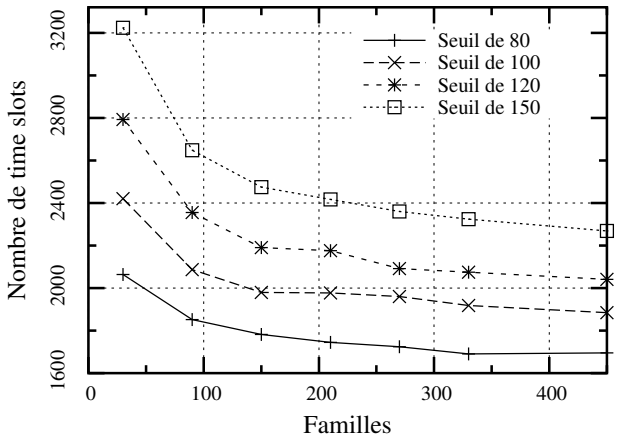


FIG. 5.13 – Aires obtenues pour différents nombres de familles aléatoires.

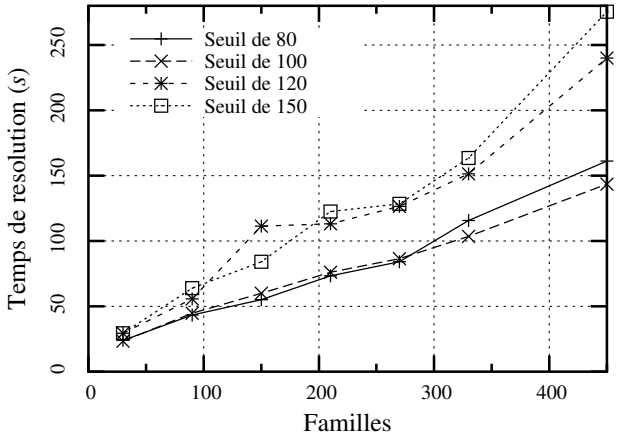


FIG. 5.14 – Temps de calcul obtenus pour différents nombres de familles aléatoires.

programme linéaire, nous avons montré comment le résoudre efficacement en utilisant la génération de colonnes et un système maître/esclave. Cet algorithme, à partir de différentes familles typées, permet de converger vers plusieurs solutions optimales fractionnaires. À partir de ces solutions nous arrivons à extraire une solution entière en se limitant à une distance de 1% de la solution optimale.

À partir du résultat calculé par notre système maître/esclave, nous avons précisé comment réaliser le placement des différents time-slots sur la bande de fréquence. Les résultats expérimentaux permettent alors de représenter le plan de fréquence du satellite. Par ailleurs, nos algorithmes permettent de déterminer les familles de spots qui ne deviennent plus valides quand l'interférence augmente, ce qui se révèle très utile du point de vue de l'opérateur.

Conclusion et perspectives

La première partie de cette thèse s'est penchée sur trois types de problèmes d'allocation de ressources dans les réseaux optiques ainsi que sur la résolution complète de l'allocation d'un plan de fréquences dans les réseaux satellitaires.

L'étude du routage a permis d'introduire plusieurs formes d'expression du multiflot et nous a conduit vers l'introduction d'une approximation du multiflot fractionnaire. Les améliorations apportées permettent d'obtenir de meilleures performances lorsqu'il s'agit de réaliser des calculs intensifs, ce qui se révèle particulièrement utile pour des réseaux de grande taille. La problématique du groupage nous a conduit à élaborer plusieurs heuristiques qui se basent sur les algorithmes de routage précédents, démontrant l'influence et l'importance de la phase de routage avant le groupage proprement dit. Ces heuristiques montrent qu'il est possible de router des connexions en gaspillant des ressources pour grouper ensuite ces connexions dans des conteneurs de plus haut niveau qui ont un coût d'acheminement inférieur dans le réseau.

Nous envisageons par la suite de poursuivre l'étude du groupage en introduisant la relaxation lagrangienne comme un moyen de traverser le polyèdre des solutions d'un programme linéaire adapté à ce problème. L'expression d'un tel programme linéaire pourrait se servir du modèle en tubes et chercherait à minimiser le nombre de tubes utilisés pour l'acheminement du routage précalculé.

Le chapitre consacré à la protection présente un modèle détaillé permettant de calculer un ensemble de chemins principaux et de secours, garantissant la possibilité de protéger 100% des pannes possibles. À partir de ce modèle, nous présentons les améliorations que nous apportons et qui visent l'obtention de résultats sur des réseaux optiques de taille raisonnable. La méthode présentée permet de relâcher la contrainte d'intégrité du modèle et d'appliquer ensuite un algorithme d'arrondis aléatoire, ce qui permet de retrouver une solution entière. Des garanties théoriques sont obtenues quant à la proximité de la solution arrondie, par rapport à la solution entière exacte. La qualité de l'ensemble de la méthode ne dépend alors que de l'ensemble des chemins candidats pour lesquels nos deux heuristiques de calculs ont été développées et expérimentées.

Nous souhaitons aborder le problème de la protection en élargissant le domaine : nos prochains travaux cherchent à prendre en compte la protection sur deux couches réseaux en même temps : la couche paquet et la couche d'agrégation (ou couche de flux), comme par exemple IP/WDM. L'objectif d'une telle étude est de prévoir la coopération des deux systèmes de protection pour éviter de protéger deux fois, une fois au niveau WDM et une fois au niveau IP, un même ensemble de requêtes.

Enfin, le dernier chapitre de cette partie a présenté une méthodologie complète pour l'allocation d'un plan de fréquences sous contraintes d'interférences pour un réseau satellitaire. La méthode proposée se base sur un programme linéaire mixte dont la résolution fait appel

à des itérations successives d'un programme maître et d'un programme esclave. Les expérimentations permettent alors d'exhiber les configurations de fréquences qui sont perdues avec l'augmentation du seuil d'interférence.

La deuxième partie de cette thèse présente les développements logiciels spécifiques issus de nos travaux sur les réseaux optiques et qui ont permis de réaliser nos expérimentations.

Deuxième partie

Contributions logicielles pour la conception de réseaux

Introduction

La deuxième partie de cette thèse présente les contributions logicielles qui ont été apportées au cours de l'implémentation pratique des méthodes présentées en première partie. Les algorithmes développés dans cette thèse visant à répondre à des problèmes d'allocation de ressources dans des réseaux réels, l'implémentation pratique de nos méthodes étaient donc primordiales. La réalisation dépasse la simple validation d'un principe et résulte en la production de modèles et d'outils performants pour qu'un chercheur puisse programmer ses expérimentations plus facilement.

Ce type d'étude est motivé par le fait que le passage de la théorie à la pratique n'est pas trivial. Les algorithmes qui peuvent avoir une complexité asymptotiquement bonne ne sont pas forcément efficaces, rencontrant des problèmes d'implémentation liés par exemple aux langages, à la gestion de l'espace mémoire et des données. Ainsi, nous détaillons dans la suite les travaux logiciels de cette thèse qui accompagnent la première partie théorique.

Cette partie décrit deux contributions précises : le logiciel PORTO, outil d'optimisation de ressources dans les réseaux optiques, et la bibliothèque MASCOPT, implémentation d'un modèle algorithmique de graphes et de réseaux.

Le chapitre 6 décrit PORTO, qui est le fruit d'un projet RNRT, partenariat entre ALCATEL RESEARCH & INNOVATION, FRANCE TÉLÉCOM et MASCOTTE. Ce logiciel concrétise la vision d'un opérateur qui souhaite allouer pour ses clients les ressources de son réseau WDM. Nous implémentons ainsi un certain nombre d'algorithmes présentés dans la première partie et nous décrivons les choix d'implémentation qui ont permis de réaliser cet outil.

À partir de ce premier logiciel, nous décrivons dans le chapitre 7 comment notre expérience acquise lors du projet PORTO nous a permis de définir un modèle pour les graphes et les réseaux plus général que celui pour les réseaux optiques. Nous présentons alors les objectifs de la bibliothèque MASCOPT et l'architecture logicielle de celle-ci en montrant comment les choix d'implémentation réalisés répondent à nos problématiques de réseaux.

CHAPITRE 6

Porto

Le logiciel PORTO [**IMR01**] (pour **P***lanification et* **O***ptimisation des* **R***éseaux de* **T***ransport* **O***ptiques*) est le fruit d'un projet RNRT entre trois partenaires : ALCATEL RESEARCH & INNOVATION, FRANCE TÉLÉCOM R&D et MASCOTTE. Il s'agissait de développer une plate-forme logicielle qui vise à optimiser les coûts de dimensionnement des réseaux optiques WDM. En entrée, on donne un réseau physique (nœuds et câbles) et une matrice de trafic à réaliser (exprimée en trames STM fournie par FRANCE TÉLÉCOM). Les modules d'optimisation calculent l'allocation des conteneurs optiques (fibres, bandes ou longueurs d'onde) des requêtes en minimisant une fonction de coût associée au nœuds du réseau (donnée ici par Alcatel). La difficulté du problème consiste à grouper au maximum les requêtes partageant des chemins physiques communs pour utiliser les brasseurs de plus faibles coûts possibles (dans l'ordre : fibre, bande et longueur d'onde). Les méthodes de résolution utilisées sont présentés dans les autres chapitres de cette thèse et sont principalement basées sur l'expression du flot au travers de la programmation linéaire en nombres entiers combinée à des heuristiques.

Le logiciel PORTO a fait intervenir de nombreux développeurs au sein de MASCOTTE. Le travail d'implémentation de cette thèse a consisté à mettre en place les différents modules de routage et de protection, notamment en les interfaçant avec ILOG CPLEX SOLVER™. À partir de ces modules et des autres algorithmes tels que le groupage, un travail d'expérimentation a permis de réaliser la validation pratique de certaines heuristiques, en particulier celles du chapitre 3 sur le groupage.

Par la suite, nous présentons le modèle de données de PORTO. Il s'agit d'expliciter comment, à partir du modèle mathématique du chapitre 1, nous implémentons réellement les données au travers d'un fil d'optimisation. Nous montrons ainsi en quoi le format proposé est adapté aux modules de routage, groupage et de protection de l'outil. Nous décrivons ensuite brièvement les algorithmes d'optimisation de PORTO et nous présentons en dernière partie les résultats obtenus sur les trois réseaux d'essai fournis par FRANCE TÉLÉCOM, dont deux ne sont pas publics. Notons enfin qu'en annexe B, nous décrivons les choix d'implémentation et l'interface du logiciel et nous présentons le déroulement d'une séquence d'utilisation dans cette interface.

6.1. Les données du problème traité dans Porto

Le problème traité dans PORTO a été posé par FRANCE TÉLÉCOM et est rappelé dans le document final de synthèse [**IMR01**]. Ce problème reprend un certain nombre de thématiques présentées dans cette thèse. Globalement, PORTO doit aider l'opérateur à allouer des ressources sur un réseau optique à trois niveaux de multiplexage (fibre, bande, longueurs d'onde) à l'aide

d'algorithmes à élaborer. FRANCE TÉLÉCOM et ALCATEL RESEARCH & INNOVATION disposaient de plusieurs logiciels répondant partiellement au problème et jugés non satisfaisants. Ce projet RNRT s'est donc révélé comme une opportunité pour MASCOTTE d'étudier des problèmes réels et non pas construits à partir de réseaux artificiels.

Pour faire le lien avec les problèmes déjà traités dans cette thèse, nous détaillons les attentes de FRANCE TÉLÉCOM et ALCATEL RESEARCH & INNOVATION, quant aux fonctionnalités de PORTO :

– Réaliser un modèle hiérarchique à trois niveaux de réseaux optiques.
– Grouper les longueurs d'onde à l'intérieur des trois niveaux d'encapsulation du modèle.
– Router des demandes exprimées en trames STM sur ce réseau.
– Permettre la sécurisation des requêtes sur le réseau.

Nous ne rappelons pas tous les éléments théoriques cités dans les trois premiers chapitres qui répondent à ces différents points. Nous nous contentons de préciser les points spécifiques concernant PORTO.

Le modèle hiérarchique à trois niveaux est présenté dans le chapitre 1, en section 1.1. L'originalité de ce modèle se trouve dans l'utilisation des bandes et des brasseurs de bandes, les B-OXC, qui à l'époque de la réalisation de PORTO n'étaient généralement pas pris en compte dans les modèles de la littérature. En fait, les modèles optiques étaient la plupart du temps bi-couche (des longueurs d'onde dans des fibres) et nous avons proposé de le généraliser à un modèle multi-couche. Dans nos expérimentations nous nous limitons à deux ou trois couches. Considérer trois niveaux hiérarchiques plutôt que deux nous a poussé à développer des modèles plus génériques qui permettent de considérer des réseaux incluant d'autres couches comme SDH ou IP.

Les algorithmes de routage utilisés dans PORTO sont issus des modèles présentés au chapitre 2. La section 6.3 précise les particularités de ces routages dans l'implémentation de PORTO. De même, les algorithmes de groupage font référence au chapitre 3.

Enfin, la sécurisation a fait l'objet d'un algorithme de bi-routage, permettant de réaliser une protection de type $1 + 1$, comme expliqué au chapitre 4, en section 4.2.2.

6.2. Architecture logicielle

Cette section décrit les choix d'implémentation réalisé dans PORTO. Ces choix découlent à la fois de nos besoins et des contraintes pratiques du processus de développement. PORTO a été réalisé dans le cadre d'un contrat RNRT sur une durée limitée à trois ans, l'outil devant être délivré à nos partenaires pour sa validation au terme du contrat.

6.2.1. Découpage fonctionnel

Du point de vue organisation logicielle, on distingue trois entités, présentées dans la figure 6.1 : les modules de calcul de routage et de groupage, l'interface et les données. Le module de calcul et l'interface graphique de l'outil PORTO communiquent à travers le fichier XML modélisant les données, présenté en section 6.4. La structure de données XML décrit une instance valide du réseau dimensionné ou non : cette structure contient aussi bien la description du réseau physique et de ses caractéristiques que les demandes et l'affectation de ces demandes aux ressources du réseau. Ce choix peut sembler limiter les interactions entre les deux parties mais il n'y a en fait pas d'affichage prévu sur la vue réseau au cours des calculs.

La figure 6.1 qui montre la séparation des calculs et de l'interface autour du modèle de données précise que l'utilisateur peut agir manuellement sur le modèle : il peut par exemple

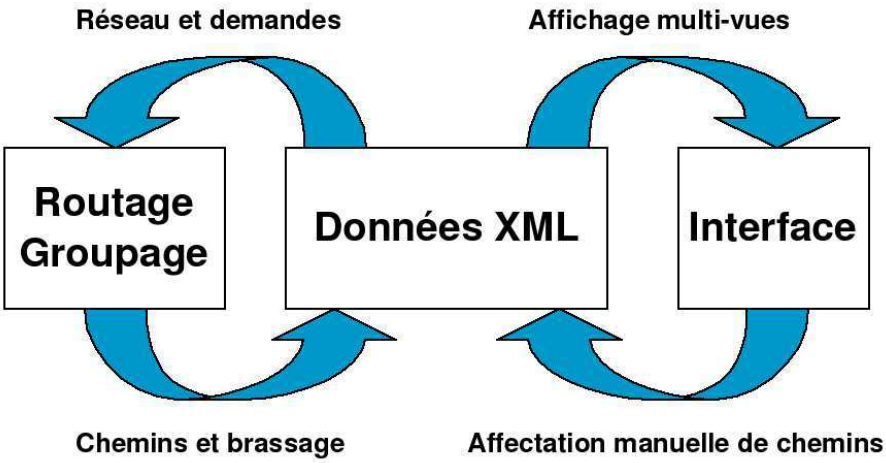


FIG. 6.1 – Découpage fonctionnel de l'outil PORTO

définir lui-même des routes sur le réseau, s'il souhaite introduire avant un calcul des routes fixes dans son réseau.

Le processus d'optimisation obtenu est donc itératif : il permet de réaliser une succession de calculs de routes et de groupages avec validation ou non de chaque calcul (s'il n'est pas jugé satisfaisant). Entre chaque itération, des affichages ou des traitements externes (par exemple, la modification des données) peuvent être introduits.

Nous n'avons malheureusement pas eu le temps d'implémenter et de tester autant d'heuristiques que souhaité dans la durée limitée du contrat. Cependant, dans l'optique d'une poursuite des travaux d'expérimentations dans PORTO, nous avons implanté un système de "plugin" permettant d'ajouter de nouvelles heuristiques plus facilement.

6.2.2. Un système de plugin

Nous introduisons ici la notion de "plugin", c'est à dire un module de programmation pouvant être ajouté à PORTO relativement facilement. La notion de plugin est particulièrement importante lorsqu'on souhaite faire intervenir sur un logiciel d'autres partenaires qui souhaitent implanter de nouveaux algorithmes.

Dans le code source 6.1, on peut observer l'arborescence des fichiers de la distribution de PORTO. Nous ne détaillons pas le fonctionnement des classes de cette arborescence mais préférons expliquer comment un utilisateur peut implanter un nouvel algorithme de routage ou de groupage.

Prenons le cas d'un algorithme de routage. Tous les algorithmes disponibles à travers l'interface sont rassemblés dans le répertoire *src/routing_algorithms/* et utilisent un module commun de calcul des chemins dont le fichier source est stocké dans *src/routing_engine/*. Imaginons que l'algorithme consiste à router seulement une partie des demandes ayant une taille minimum donnée en paramètre (pour faire par exemple une optimisation en plusieurs étapes). Il suffit alors d'écrire le programme C++ qui appelle *routing_engine* avec en paramètre l'ensemble des demandes filtrées. Lors de la prochaine exécution de PORTO cet algorithme sera disponible et automatiquement intégré à l'interface.

L'intérêt d'un tel système est de fournir, à partir d'une connaissance minimale des classes de PORTO, la possibilité d'écrire de nouveaux algorithmes d'optimisation qui réutilisent les

données mais aussi les autres algorithmes de PORTO. Ainsi, nous appelons “plugin” un nouveau programme C++ ayant une interface convenue à l’avance et s’intégrant à l’outil. Chaque plugin est en fait un algorithme, ou un algorithme de plus haut niveau utilisant d’autres plugins (ie. un algorithme utilisant des sous-algorithmes).

CODE SOURCE 6.1 – Arborescence de PORTO

```
./docs
./log
./net
./net/networks
./plot
./script
./script/all_algos
./script/filter
./script/load
./src
./src/engine
./src/plugin_algorithms
./src/plugin_engines
./src/viewer
./xml
./xml/networks
```

6.3. Algorithmes implémentés dans PORTO

Nous décrivons brièvement la spécificité des algorithmes implémentés dans PORTO, à partir des méthodes de résolution proposés dans cette thèse. Ces algorithmes peuvent être librement composés par l’utilisateur : il peut par exemple réaliser un routage, puis un groupage et même un routage partiel suivi du groupage de ces routes et réitérer cette séquence jusqu’au traitement de toutes les requêtes. La protection se substitue au routage classique : ce module est considéré comme un routage particulier.

6.3.1. Routage

PORTO utilise la formulation *sommet-arc* du multiflot, à savoir le programme linéaire 3 de la section 2.1.2. Ce module de routage dénommé *multi* est paramétrable : on peut préciser l’ensemble des requêtes à considérer en les discriminant par tailles. Un algorithme glouton de routage *order* découle de l’implémentation de ce dernier : on route les requêtes une à une par ordre croissant ou décroissant de taille.

Un deuxième programme linéaire permet de réaliser un routage monoflot. Ce module utilise le programme linéaire 5 de la section 4.2.2 et est dénommé dans PORTO *single*. Là aussi, il peut être utilisé pour l’algorithme de routage glouton *order*.

6.3.2. Groupage

La phase de groupage utilise l’algorithme de groupage local de la section 3.3.2. Là aussi, des variantes permettent de paramétrer ce groupage local, en ne groupant que certaines requêtes (classées par groupe de taille par exemple).

À partir de ce groupage local, les algorithmes de la section 3.4 on été implémentés dans PORTO. Ils illustrent comment mettre à profit l’architecture logicielle de PORTO pour interfacer les différents algorithmes. Dans les résultats présentés dans la section 3.4 on montre comment le routage influe sur le groupage. Au niveau de l’implémentation, nous avons défini des modules

de plus haut niveau qui permettent d'appeler le module de routage, puis le module de groupage et d'itérer jusqu'à router et grouper l'ensemble du trafic.

Notons encore que ces algorithmes de groupages sont génériques, par rapport au nombre de niveaux hiérarchiques considérés. PORTO permet de réaliser tous ces calculs pour un nombre de niveaux quelconque. Les résultats expérimentaux présentent d'ailleurs les résultats pour les réseaux à deux et trois niveaux.

6.3.3. Protection

La protection est implémentée par un module de routage particulier. Ce module utilise l'expression du "bi-routage" du programme linéaire 4.22. Etant donné la complexité du problème, ce module ne donne pas toujours en un temps raisonnable une solution (ce qui s'observe dans la partie consacrée au résultats expérimentaux, en section 6.5).

6.3.4. Fonction objectif

La fonction objectif utilisée dépend de chaque module, et donc de chaque algorithme. Comme précisé dans les chapitres 2 et 3, on n'utilise pas le même critère pour le problème du routage et du groupage. Ainsi, lors du routage, PORTO minimise la charge totale alors que pour le groupage on cherche plutôt à minimiser le nombre de ports des multiplexeurs.

Ces fonctions objectifs sont corrélées à une autre valeur : celle du coût du réseau vu par l'équipementier partenaire du projet : ALCATEL RESEARCH & INNOVATION. PORTO permet de réaliser ce calcul, a posteriori du routage et du groupage. On peut ainsi évaluer le coût de l'allocation des ressources du réseau, selon une formule non linéaire dépendante de l'occupation des câbles et de la taille des multiplexeurs au niveau des nœuds.

Notons enfin que le coût du réseau est différent de la fonction objectif utilisée. Le coût, donné par ALCATEL RESEARCH & INNOVATION, est une fonction non linéaire difficile à utiliser dans nos programmes linéaires. Par ailleurs, les étapes d'optimisation étant séparées, il n'est pas forcément possible de calculer le coût lorsque seulement une partie des calculs a été réalisée.

6.3.5. Autres plugins

D'autres plugins ont été développés dans PORTO autour des principaux algorithmes d'optimisation.

Au niveau des données d'entrée, des générateurs de réseaux (aléatoires, grilles, tores, cycles) permettent de tester des topologies régulières. Sur ces réseaux, des générateurs de trafic (aléatoire, *All-to-All*, *All-to-Many*,...) permettent d'instancier les requêtes sur ces réseaux. Certains plugins permettent de modifier un réseau (suppression, ajout d'un ou plusieurs câbles) ou son instance de communication (augmentation de la taille ou du nombre de requêtes).

Les données de sortie de PORTO peuvent elles aussi recevoir des traitements particuliers. Il s'agit en général de réaliser des graphiques en faisant varier un des paramètres d'entrée et en mesurant une des valeurs de sortie (coût, nombre de longueurs d'onde groupées, taille des équipements des nœuds). Les plugins développés à cet effet permettent de construire l'expérimentation et de collecter les données dans un fichier GNUPLOT permettant de générer les graphiques correspondants.

6.4. Modèle de données détaillé de PORTO

Cette section présente le modèle XML (*eXtended Markup Language*) des données de PORTO. Ce modèle, vu ici à travers un format de fichier, permet d'aller plus loin que la simple sauvegarde des données dans un fichier : il représente un modèle descriptif d'un réseau optique et des données qui y circulent.

6.4.1. Origine des données et formalisation

Outre les études algorithmiques des problèmes de routage, groupage et protection, largement abordés précédemment, une première phase de l'analyse du problème a consisté à formaliser les données fournies par nos partenaires. La formalisation mathématique de ces données est proche de celle décrite au chapitre 1 mais la représentation physique de ces données n'est pas triviale et dépend fortement de l'architecture logicielle envisagée. Dans PORTO, notre souci était de pouvoir décrire le réseau, les communications qui le traversent, c'est-à-dire le routage et le groupage, ainsi que la sécurisation, et ceci de manière partielle (par exemple, décrire un routage sans groupage, ou un routage partiellement fait). Par ailleurs, les données brutes décrivant les réseaux et les requêtes nous étant fournies sous la forme de fichiers au formats variables MICROSOFT™ EXCEL™, elles n'étaient pas directement exploitables et réutilisables par un programme.

Nous avons donc défini une grammaire XML permettant d'écrire des fichiers et répondant aux besoins précédemment exprimés. Ce format a notamment permis d'échanger des fichiers entres différents programmes et différents partenaires. La problématique à traiter s'intéressant principalement au groupement des longueurs d'onde dans les liens du réseau, le format de données décrit ce réseau arc par arc, en précisant récursivement les différents conteneurs qui le composent. Plus précisément, nous définissons la hiérarchie suivante :

(1) Au plus haut niveau se trouve un câble, représenté par la balise `<CABLE></CABLE>`

(2) À l'intérieur, on trouve une ou plusieurs fibres, représentées par des balises `<FIBER></FIBER>`

(3) De même, on trouve ensuite une ou plusieurs bandes, représentées par des balises `<BAND></BAND>`

(4) De même, on trouve ensuite une ou longueurs d'onde, représentées par des balises `<LAMBDA></LAMBDA>`

CODE SOURCE 6.2 – Partie d'un fichier XML de PORTO

```
<CABLE from="N_0" to="N_1" id="CABLE_0_1" length="243">
        <FIBER id="FIBER_0_1_0">
                <BAND id="BAND_0_1_0_0">
                        <LAMBDA id="LAMBDA_0_1_0_0_0"/>
                        <LAMBDA id="LAMBDA_0_1_0_0_1"/>
                        <LAMBDA id="LAMBDA_0_1_0_0_2"/>
                        <LAMBDA id="LAMBDA_0_1_0_0_3"/>
                        <LAMBDA id="LAMBDA_0_1_0_0_4"/>
                        <LAMBDA id="LAMBDA_0_1_0_0_5"/>
                        <LAMBDA id="LAMBDA_0_1_0_0_6"/>
                        <LAMBDA id="LAMBDA_0_1_0_0_7"/>
                </BAND>
                <BAND id="BAND_0_1_0_1">
...
                </BAND>
```

```
        </FIBER>
</CABLE>
<CABLE from="N_1" to="N_0" id="CABLE_1_0" length="243">
        <FIBER id="FIBER_1_0_0">
                <BAND id="BAND_1_0_0_0">
                        <LAMBDA id="LAMBDA_1_0_0_0_0"/>
    ...
```

Un exemple de cette représentation est donnée dans le code source 6.2. Cette vision du réseau suppose que les nœuds sont définis avant les câbles. En fait, on identifie simplement les nœuds par des "ids" qui permettent ensuite de raccrocher chaque extrémité du câble. La déclaration de trois nœuds est faite dans le code source 6.3.

CODE SOURCE 6.3 – Définition des nœuds

```
<NODE id="N_0" name="Lille" x="464" y="41"/>
<NODE id="N_1" name="Paris" x="444" y="172"/>
<NODE id="N_2" name="Reims" x="525" y="132"/>
```

Notons que la codification des identifiants des longueurs d'onde (LAMBDA) permet de retrouver l'emplacement de cette longueur d'onde dans un câble du réseau. Pour un identifiant tel que "LAMBDA_x_y_f_b_l", on sait que cette longueur d'onde part du nœud x pour arriver au nœud y en empruntant la fibre f, la bande b et la longueur d'onde numéro l.

6.4.2. Description des requêtes dans le modèle de données

Les requêtes à traiter sont simplement définies à la suite de la description du réseau. De manière similaire à la description des nœuds, un système d'identifiant permet de déclarer ces requêtes en les raccrochant aux nœuds du réseau. Un exemple déclarant quelques requêtes est présenté dans le code source 6.4. Dans cet exemple, l'attribut "value" permet de préciser la quantité de longueurs d'onde à router, par exemple 21 longueurs d'onde du nœud 0 au nœud 1.

CODE SOURCE 6.4 – Définition des requêtes

```
<REQUEST id="R_0" from="N_0" to="N_1" value="21"/>
<REQUEST id="R_1" from="N_1" to="N_0" value="21"/>
<REQUEST id="R_2" from="N_0" to="N_2" value="2"/>
```

6.4.3. Intégration de l'allocation des ressources

Le modèle de données ainsi défini permet de renseigner les informations issues des algorithmes développés, que ce soit le routage, le groupage ou la protection.

6.4.3.1. *Le routage.* Pour définir le routage des requêtes, que ce soit au niveau des routes sur la vue topologique du réseau ou au niveau des longueurs d'onde utilisées pour une vue d'affectation des longueurs d'onde, on utilise de nouveau la hiérarchie XML définie précédemment. Au niveau de la balise "<LAMBDA></LAMBDA>", on introduit la balise "<ROUTE></ROUTE>" définissant qu'une route ayant un certain identifiant utilise cette longueur d'onde. Avec un tel modèle il est alors facile de savoir si une longueur d'onde est libre ou non. Le code source 6.5 montre comment les routes sont créées à partir du réseau vide du code source 6.2.

Il faut noter, dans cet exemple, qu'une route ayant un même identifiant (par exemple, la route P_40) puisse apparaître sur plusieurs longueurs d'onde. Une route est un chemin dans le graphe sous-jacent. Dans un câble, on trouve autant de longueurs d'onde appartenant à cette

route que la taille de la requête transportée. On peut donc trouver, dans un même câble, plusieurs fois l'identifiant de cette route, pour chaque longueur d'onde qu'elle véhicule.

CODE SOURCE 6.5 – Définition des routes

```
<CABLE id="CABLE_0_1" length="243" from="N_0" to="N_1">
        <FIBER id="FIBER_0_1_0" inputState="close" prev="add" outputState="open
            ">
                <BAND id="BAND_0_1_0_0" outputState="close" next="drop">
                        <LAMBDA id="LAMBDA_0_1_0_0_0">
                                <ROUTE pathId="P_40"/>
                        </LAMBDA>
                        <LAMBDA id="LAMBDA_0_1_0_0_1">
                                <ROUTE pathId="P_40"/>
                        </LAMBDA>
                        <LAMBDA id="LAMBDA_0_1_0_0_2">
                                <ROUTE pathId="P_40"/>
                        </LAMBDA>
                        <LAMBDA id="LAMBDA_0_1_0_0_3">
                                <ROUTE pathId="P_40"/>
                        </LAMBDA>
                        <LAMBDA id="LAMBDA_0_1_0_0_4">
                                <ROUTE pathId="P_41"/>
                        </LAMBDA>
                        <LAMBDA id="LAMBDA_0_1_0_0_5">
                                <ROUTE pathId="P_41"/>
                        </LAMBDA>
                        <LAMBDA id="LAMBDA_0_1_0_0_6">
                                <ROUTE pathId="P_41"/>
                        </LAMBDA>
                        <LAMBDA id="LAMBDA_0_1_0_0_7">
                                <ROUTE pathId="P_42"/>
                        </LAMBDA>
                </BAND>
    ...
```

6.4.3.2. *Le groupage.* La modélisation du groupage intervient au niveau des attributs d'un conteneur d'un niveau donné. Chaque conteneur possède un attribut "prev" et "next" définissant le conteneur suivant lorsque une route est créée (ce qui explique pourquoi ces attributs ne sont pas définis dans le code source 6.2 mais le sont dans le code source 6.5). La signification de la valeur des attributs "prev" et "next" sont définis ci-dessous :

- **add** : le conteneur est ajouté au nœud source de ce câble.
- **drop** : le conteneur est supprimé au nœud destination de ce câble.
- **CONTENEUR_X** : le conteneur de ce niveau traverse le nœud destination vers un autre câble.

Cette dernière information précise directement quel est le groupage du trafic. En effet, si c'est le conteneur "LAMBDA" qui possède l'attribut "next" initialisé à un identifiant d'un autre conteneur "LAMBDA", alors le multiplexage au niveau du brasseur se fait au niveau W-OXC : les longueurs d'onde sont brassées au niveau le plus bas (pas de groupage). Si c'est le conteneur "BAND" qui possède l'attribut "next" initialisé à un identifiant d'un autre conteneur "BAND", alors le multiplexage se fait au niveau B-OXC ; de même pour le niveau fibre. Nous donnons un exemple de groupage d'une fibre qui traverse directement le nœud dans le code source 6.6.

CODE SOURCE 6.6 – Groupage et attributs prev et next

```
<CABLE id="CABLE_1_2" length="167" from="N_1" to="N_2">
```

```
<FIBER id="FIBER_1_2_0" inputState="close" prev="add" outputState="
    close" next="FIBER_2_3_0">
        <BAND id="BAND_1_2_0_0">
                <LAMBDA id="LAMBDA_1_2_0_0_0">
                        <ROUTE pathId="P_0"/>
...
```

Suivant le devenir du conteneur dans ce nœud, on précise l'état du conteneur en entrée et en sortie par l'intermédiaire des attributs "inputState" et "outputState". Si l'état est à `close`, le groupage a été réalisé au niveau de ce nœud et le conteneur traverse directement l'OXC de son niveau. Si l'état est à `open` l'acheminement de ce conteneur est interrompu pour être soit démultiplexé puis remultiplexé dans un autre conteneur, soit supprimé (*drop*).

6.4.3.3. *La protection.* La gestion de la protection se fait par l'ajout d'un attribut à la balise <ROUTE>. On précise simplement son type en indiquant "type=protection" pour une route de protection et "type=principal" pour une route normale.

6.5. Extraits de résultats expérimentaux

Les résultats présentés dans cette section ont été obtenus sur les réseaux R_1, R_2 et R_3 avec les hypothèses de 32 longueurs d'onde par fibres groupées en 4 bandes de 8 longueurs d'onde chacune (les réseaux sont décrits en annexe A.2).

La table 6.1 résume les résultats obtenus pour le réseau R_3 avec les algorithmes de routage suivants : *multi* sans protection, *single* sans protection et *single* avec protection des demandes en $1{+}1$. Le groupage est celui obtenu par l'heuristique de groupage local, présenté en section 3.3.2 avec l'option "by_size" de l'interface graphique : cette heuristique parcourt les requêtes dans l'ordre croissant de leur taille.

La première partie du tableau correspond aux résultats obtenus pour les trois niveaux de granularités, fibres, bandes et longueurs d'onde. La deuxième partie du tableau correspond aux résultats pour des essais à deux niveaux seulement, fibres et longueurs d'onde. Les chiffres présentés correspondent au nombre de fibres fermées obtenues, au nombre de bandes fermées (sauf pour le cas du réseau à deux niveaux) et au nombre de longueurs d'onde non incluses dans des bandes ou fibres fermées. La dernière ligne indique le nombre total de longueurs d'onde utilisées dans le réseau, c'est-à-dire la somme des longueurs des chemins de longueurs d'onde.

La première valeur indiquée pour ce total ne prend pas en compte les *add* et *drop*, alors que toutes les autres valeurs du tableau en tiennent compte. Nous expliquons la différence entre les deux modes de comptage sur l'exemple suivant : soit un chemin de longueur 2 ($N\_0 \rightarrow N\_1 \rightarrow N\_2$) et une demande de 3 longueurs d'onde de N_0 à N_2 ; dans le premier mode on compte $3 \times 2 = 6$ longueurs d'onde utilisées ; dans le second mode on ajoute 3 longueurs d'onde *add* et 3 longueurs d'onde *drop* soit un nouveau total de 12 longueurs d'onde utilisées. Le second mode de calcul tient donc compte du groupage initial des demandes et correspond aussi à la fonction objectif programmée dans l'outil.

Les résultats obtenus pour les routages multi ou mono-chemins sans protection sont très proches car, en fait, les chemins utilisés sur ce réseau sont courts (en moyenne 1,6 arcs par chemin) et il n'y a que 113 chemins différents utilisés en multiroutage pour 110 utilisés en monoroutage. Le routage le moins contraint (multi) permet tout de même de fermer 3 fibres supplémentaires. Le routage le plus contraint (avec protection) ne permet de fermer qu'une seule fibre, car le taux d'utilisation des longueurs d'onde disponibles est alors très élevé (1250/1600) et le choix des fibres pour préserver le groupage devient quasi nul.

	trois niveaux d'encapsulation		
	multiroutage sans protection	monoroutage sans protection	monoroutage protection 1+1
Nombre de fibres fermées	25	22	×
Nombre de bandes fermées	40	42	×
Nombre de longueurs d'onde	421	420	×
Total (longueurs d'onde)	514/1214	514/1214	×
	deux niveaux d'encapsulation		
Nombre de fibres fermées	25	22	×
Nombre de longueurs d'onde	711	707	×
Total (longueurs d'onde)	514/1214	514/1214	×

TAB. 6.1 – Résultat de l'allocation pour le réseau R_3

	trois niveaux d'encapsulation		
	multiroutage sans protection	monoroutage sans protection	monoroutage protection 1+1
Nombre de fibres fermées	41	38	×
Nombre de bandes fermées	77	77	×
Nombre de longueurs d'onde	624	647	×
Total (longueurs d'onde)	1136/2520	1136/2520	×
	deux niveaux d'encapsulation		
Nombre de fibres fermées	41	38	×
Nombre de longueurs d'onde	1120	1229	×
Total (longueurs d'onde)	1136/2520	1136/2520	×

TAB. 6.2 – Résultat de l'allocation pour le réseau R_1

La table 6.2 résume les résultats pour le réseau R_1 avec les deux algorithmes de routage sans protection. Nous observons des résultats similaires à ceux déjà commentés pour le réseau R_3, à savoir qu'il y a peu de différence entre les deux types de routage sans protection. En revanche, la contrainte de protection entraîne ici un échec. En effet, le taux d'utilisation des longueurs d'onde est supérieur à celui de R_3, si on double simplement la valeur de 1136 du routage sans protection (pour estimer le total non calculé), on obtient un taux de 2272/2752, soit environ 82,6% (contre 64,25% dans le cas de R_1). De plus, les chemins utilisés dans le réseau R_1 sont supérieurs d'un arc en moyenne, ce qui augmente la complexité du routage arcs disjoints des chemins principaux et des chemins de protection.

Les résultats obtenus pour le réseau R_2 sont résumés dans la table 6.3, avec les mêmes hypothèses que pour le réseau précédent. Peu de bandes sont fermées car le nombre de longueurs d'onde utilisées par les routages est faible par rapport au nombre total disponible de 4928. Ainsi on peut utiliser de nombreuses fibres libres, même si les demandes ne les remplissent pas complètement. On observe que lorsque le taux d'utilisation devient plus important (1152/4928) pour le cas du routage avec protection, le nombre de fibres fermées est diminué par 3 et les algorithmes de groupage ferment 10 fois plus de bandes.

	trois niveaux d'encapsulation		
	multiroutage sans protection	monoroutage sans protection	monoroutage protection 1+1
Nombre de fibres fermées	160	157	45
Nombre de bandes fermées	5	5	54
Nombre de longueurs d'onde	204	200	1163
Total (longueurs d'onde)	368/992	368/992	1152/1776
	deux niveaux d'encapsulation		
Nombre de fibres fermées	160	157	45
Nombre de longueurs d'onde	237	233	1527
Total (longueurs d'onde)	368/992	368/992	1152/1776

TAB. 6.3 – Résultat de l'allocation pour le réseau R_2

Conclusion

Cette première réalisation logicielle montre l'intégration des algorithmes présentés dans la première partie de cette thèse dans un outil concret de planification de réseaux optiques à multi-niveaux. Cet outil est directement exploitable par un opérateur et nous avons montré les méthodes mises en œuvre pour faciliter son enrichissement.

En réalité, il y a eu quelques problèmes dans l'exploitation effective de ce logiciel. Certains sont dus aux difficultés de communication entre les partenaires du projet, notamment au niveau logiciel : expertises et cultures différentes. Enfin, malgré les efforts de chacun il y a aussi eu des problèmes pour faire évoluer le logiciel, même au sein de MASCOTTE et pour expérimenter de nouveaux algorithmes. Ces problèmes sont issus de la difficulté à gérer différentes versions des bibliothèques utilisées et à la nécessité de compétences pointues chez les utilisateurs potentiels.

Le chapitre qui suit présente une deuxième réalisation logicielle : MASCOPT. En particulier, nous essaierons d'introduire les réflexions issues de PORTO et qui ont conduit à la réalisation de MASCOPT.

CHAPITRE 7

Mascopt

MASCOPT [1] est une bibliothèque d'optimisation pour les graphes et les réseaux. Son développement a débuté en même temps que cette thèse et se poursuit aujourd'hui encore. La bibliothèque MASCOPT a permis de tester un certain nombre d'algorithmes présentés dans la première partie mais a aussi constitué un enjeu à elle seule : celui de produire une plate-forme logicielle générique permettant d'implémenter les expérimentations de l'équipe MASCOTTE ou celle de la communauté. Il ne s'agit donc plus seulement de produire un logiciel permettant d'exhiber les résultats expérimentaux de cette thèse, mais aussi d'avoir le souci de produire un code réutilisable, documenté et architecturé convenablement pour qu'il soit publiable et exploitable par autrui.

Nous présentons donc dans ce chapitre les objectifs de la bibliothèque MASCOPT, puis nous détaillons l'architecture logicielle choisie. Nous détaillons plusieurs points de l'implémentation qui font la spécificité de MASCOPT et qui répondent à la philosophie d'implémentation que nous avons choisie. Enfin, nous présentons brièvement ce qui a trait aux entrées/sorties et à l'interface graphique.

7.1. Présentation

7.1.1. L'expérience de PORTO

Le développement de MASCOPT est basé sur l'expérience acquise dans le projet RNRT PORTO [**IMR01**], présenté au chapitre 6. PORTO n'était pas, à proprement parler, une bibliothèque d'optimisation mais plutôt une application dédiée à l'optimisation des ressources des réseaux optiques à trois couches. Aussi, le développement fut axé sur le développement d'algorithmes performants répondant à des problèmes posés par FRANCE TÉLÉCOM et ALCATEL RESEARCH & INNOVATION ainsi que sur le développement d'une interface utilisateur conviviale et précise au niveau de la description du modèle. Ces exigences ne sont pas compatibles avec une politique de développement logiciel durable et générique. Le code développé n'est pas facilement réutilisable dans un autre contexte et ne permet pas, pour des développeurs n'ayant pas connaissance des choix d'implémentations de PORTO, de s'approprier le code pour en faire un usage connexe.

Sous l'impulsion de Michel Syska, un nouveau projet de développement logiciel a été initié au sein de l'équipe MASCOTTE avec des exigences dédiées au souci de pérenniser le code d'une nouvelle bibliothèque d'optimisation : MASCOPT, pour "Mascotte" et "Optimisation". Michel

[1] Ce travail a fait l'objet du rapport technique [**LSV04**]. Plus d'informations sont disponibles à `http://www-sop.inria.fr/mascotte/mascopt`.

Syska ayant été le responsable du projet PORTO au niveau du projet MASCOTTE, nous avons pu tirer parti de cette expérience dans le processus de développement de MASCOPT qui peut être difficile à maîtriser car il est collaboratif et donc sujet à tous les écueils d'une telle exigence. Les travaux effectués pendant cette thèse ont largement contribué au développement de MASCOPT qui atteint, à ce jour, près de $40\,000$ lignes de code.

7.1.2. Spécificité de Mascopt

De nombreuses bibliothèques de graphes et de réseaux existent désormais sur le marché et possèdent toutes des fonctionnalités bien identifiables. Ces bibliothèques ne reposent pas toujours sur le même objectif fonctionnel ; on peut par exemple distinguer une bibliothèque encodant un modèle de graphe au sens mathématique d'une bibliothèque de dessin de graphes. Le modèle interne est différent et les fonctionnalités proposées sont orientées par le type d'utilisation visé. Nous détaillerons donc par la suite, en décrivant les fonctionnalités de MASCOPT, la classe de problèmes qui nous intéresse.

MASCOPT est une bibliothèque *open source* déposée à l'APP[2] et dont l'objectif général est de fournir les modèles et outils nécessaires à l'implémentation d'algorithmes d'optimisation dans les réseaux. Dans ces problèmes, et comme nous le rappelons tout au long de cette thèse, les réseaux et les requêtes sur ces derniers sont en général modélisés par des graphes.

Enfin, nous omettons de parler des bibliothèques de graphes propriétaires et/ou payantes et dont le prix prohibitif des licences nous a poussé à développer MASCOPT (par exemple, ILOG Views Graph Layout[3]).

7.1.3. Une bibliothèque de manipulation de graphes

Nous présentons une brève étude comparative du marché en ce qui concerne la partie manipulation de graphe, complètement aboutie au moment de la rédaction de cette thèse. Une des bibliothèques les plus connues et qui fait référence lorsqu'on utilise le langage C++ est LEDA [**MNU97, Gmb**]. Là où LEDA vise à fournir une utilisation optimale des graphes qui sont manipulés en terme de vitesse d'exécution, MASCOPT s'intéresse beaucoup plus à l'aspect orienté objet de la manipulation de graphes. Notre souci réside dans le fait que le modèle doit rester générique et doit tirer parti d'une conception objet soigneuse pour faciliter son utilisation ultérieure. Ces considérations sont en grande partie héritées des méthodes de développement logiciel décrits dans [**Hil96**].

MASCOPT n'est pas, comme on pourrait le croire, une réécriture de LEDA en langage JAVA™. Nous montrons en section 7.2.1 les principales différences du choix d'implémentation du modèle. MASCOPT diffère aussi de la GTL (*Graph Template Library* [**FPRB**]) qui est une extension de la STL (*Standard Template Library*), bibliothèque pour les structures de données de haut niveau du C++, vers une bibliothèque générique de graphes et d'algorithmes pour les graphes. La GTL ne permet pas, par exemple, de dériver les classes de graphe facilement (les objets sont attachés aux nœuds par l'intermédiaire de maps[4] de la STL). Au niveau de la licence LEDA n'est pas libre alors que MASCOPT est publié sous licence LGPL (la GTL est pour sa part sous licence GPL).

MASCOPT est écrit en JAVA™, évitant ainsi tous les problèmes de portabilité rencontrés en C ou C++. Les performances du code obtenu sont tout à fait satisfaisantes et il est souvent

[2]Agence pour la Protection des Programmes. MASCOPT est identifié électroniquement par un numéro *Inter Deposit Digital Number at Program Protection Agency* en 2004 : IDDN.FR.001.100002.000.S.P.2004.000.31235.

[3]cf. `http://www.ilog.com/products/views/graphlayout/`

[4]table de correspondance, pouvant utiliser des tables de hachage. Typiquement, en JAVA™, une *hashmap*.

	JDSL	GFC	OJG	MASCOPT
Open Source			√	√
Graphes	√		√	√
Graphes orientés	√	√	√	√
Ensembles	√	√		√
Chemins			√	√
Parcours	√		√	√
Arbres	√		√	
Queues de priorité	√			
Tas	√			√
Système de Partage		√	√	√
Écriture fichier	√		√	√
Système de valuation				√
Visualisation		√	√	√
Éditeur			√	√
Algorithme de représentation		√	√	

FIG. 7.1 – Comparaison avec différentes bibliothèques de graphe

rapporté que l'utilisation d'un interpréteur de pseudo-code ralentit l'exécution du code alors que nos tests montrent que les temps d'exécution sont très similaires et que pour des applications conséquentes qui ont été réécrites, on constate un facteur de ralentissement allant de 2 à 4 [**HS96, Pre00**]. En fait, notre expérience montre que les performances dépendent directement des classes utilisées par le programmeur et que JAVA™, en tant que langage de haut niveau, fournit des primitives qui peuvent s'avérer être lentes car fournissant un service très complet au programmeur.

D'autres bibliothèques de manipulation de graphes ont été écrites en JAVA™ et certaines couvrent des domaines encore plus larges, comme la bibliothèque JDSL [**Ta03**] qui implémente de complexes structures de données. Nous avons étudié trois bibliothèques qui sont très proches de MASCOPT en terme de fonctionnalités. La figure 7.1 présente les fonctionnalités de JDSL, GFC [**alp**], OPENJGRAPH [**Sal**], and MASCOPT.

Comme MASCOPT est *open source* et écrit en JAVA™, il devient alors très simple de réaliser des appels vers d'autres bibliothèques externes, du *bytecode* java et même des bibliothèques compilées en C++. Nous montrons en annexe C.1 comment interfacer MASCOPT avec CPLEX SOLVER™ qui est un outil commercial très utilisé pour résoudre des programmes linéaires.

7.2. Modèles objets

7.2.1. Le modèle de graphe

Le modèle objet pour les graphes de MASCOPT est très proche de la définition mathématiques de ceux-ci. Un graphe $G = (V, E)$ est composé d'un ensemble de nœuds $V = \{v_1, \ldots, v_n\}$ et d'un ensemble d'arêtes $E = \{e_1, \ldots, e_m\}$. Chaque élément de V et de E peut être librement instancié par l'utilisateur[5], ce qui lui donne la liberté d'utiliser ces objets dans

[5]on entend par le terme "utilisateur" un programmeur qui utiliserait MASCOPT pour implémenter ses expérimentations.

d'autres structures qui lui sont propres. G est alors un graphe valide si et seulement si V et E sont cohérents : $\forall e \in E \; tq. \; e = (v_1, v_2), \; v_1 \in V, \; v_2 \in V$.

Ce modèle de graphe permet d'utiliser les possibilités fournies par la conception objet. En particulier, il n'y a pas d'objet général qui regroupe un environnement où tous les autres objets sont créés. Par exemple, on cherche à éviter un modèle où un nœud appartient à un seul et unique réseau, si l'on imagine que ce graphe représente la topologie d'un réseau. C'est cependant le choix qui a été adopté dans d'autres bibliothèques comme LEDA. Par exemple, à partir d'un graphe G, la bibliothèque LEDA crée un nœud de la façon suivante :

CODE SOURCE 7.1 – Création d'un nœud avec LEDA

```
node u = G.new_node();
```

La conséquence directe de l'accès aux constructeurs des données de base (tels que les nœuds et les arêtes) sont de deux ordres. Premièrement, l'utilisateur a donc à sa charge de créer chaque élément de V et E, puis de construire ces ensembles pour enfin construire le graphe G. Cela peut paraître fastidieux au premier abord, mais cela permet de bénéficier de la deuxième conséquence : l'utilisateur peut réutiliser des nœuds et arêtes d'un graphe $G_1 = \{V_1, E_1\}$ dans un autre $G_2 = \{V_2, E_2\}$, ceci sans les dupliquer. Nous appelons ce comportement la capacité à partager les objets et plus généralement, un "système de partage". L'utilisation de deux ensembles E_1 et E_2 n'interdit nullement d'utiliser une même arête e_i, ce qui implique que deux graphes sont différents, non pas en comparant leurs arêtes et nœuds mais bien en comparant les objets E_1 et E_2. Nous détaillons ce comportement dans la section 7.4.5.

Pour chaque type d'objet de MASCOPT (graphe, nœud, arête, ensemble, ...), nous avons factorisé les attributs et méthodes communes dans des classes abstraites, tirant ainsi partie de l'héritage (non multiple) de JAVA™. Par ailleurs, cet héritage peut se poursuivre et permettre une spécialisation de nos classes. Cela est possible pour l'utilisateur s'il crée les objets de base lui-même, par exemple, les nœuds et les arêtes : s'il spécialise ces deux classes, il pourra utiliser ses nœuds et arêtes spécialisées dans les classes des ensembles E et V sans devoir les réécrire. L'utilisateur devra seulement écrire la classe *factory* correspondante[6], en particulier lorsqu'il souhaite écrire des algorithmes génériques, ce qui est détaillé par la suite en section 7.4.4.

Nous présentons un diagramme simplifié des classes principales de MASCOPT en figure 7.2. Les diagrammes complets concernant le modèle de données de graphes se trouvent en annexe C.3. Les classes présentées sont toutes abstraites et sont brièvement présentées par la suite, ainsi que leurs interactions. L'utilisation de la notion de classes abstraites permet ensuite de dériver le comportement commun des classes de graphe vers une spécialisation choisie par l'utilisateur.

Un objet du type **AbstractGraph** représente un graphe. Cet objet nécessite un **AbstractVertexSet** et un **AbstractEdgeSet** cohérents. Un **AbstractVertexSet** (resp. **AbstractEdgeSet**) contient des nœuds (resp. arêtes). Un **AbstractPath** est un **AbstractGraph** spécialisé qui permet de construire un chemin sur un graphe.

À partir de la définition des classes abstraites de base nous fournissons deux dérivations non abstraites du modèles : la classe **Graph** qui représente un graphe non orienté et **DiGraph** qui représente un graphe orienté. Les classes **Graph** et **Digraph** n'ont pas vraiment d'implémentation propre puisque la quasi totalité du code est hérité des classes abstraites. En fait, le travail de différenciation est fait au niveau des arêtes ou des arcs, selon le cas considéré : l'arc possède une orientation et certaines méthodes de la classe **AbstractEdge** ne se comportent pas de la même façon pour les arcs. La redéfinition des classes **Graph** et **Digraph** n'apportent donc pas

[6] on peut trouver le *design pattern factory* dans [**MDE97**].

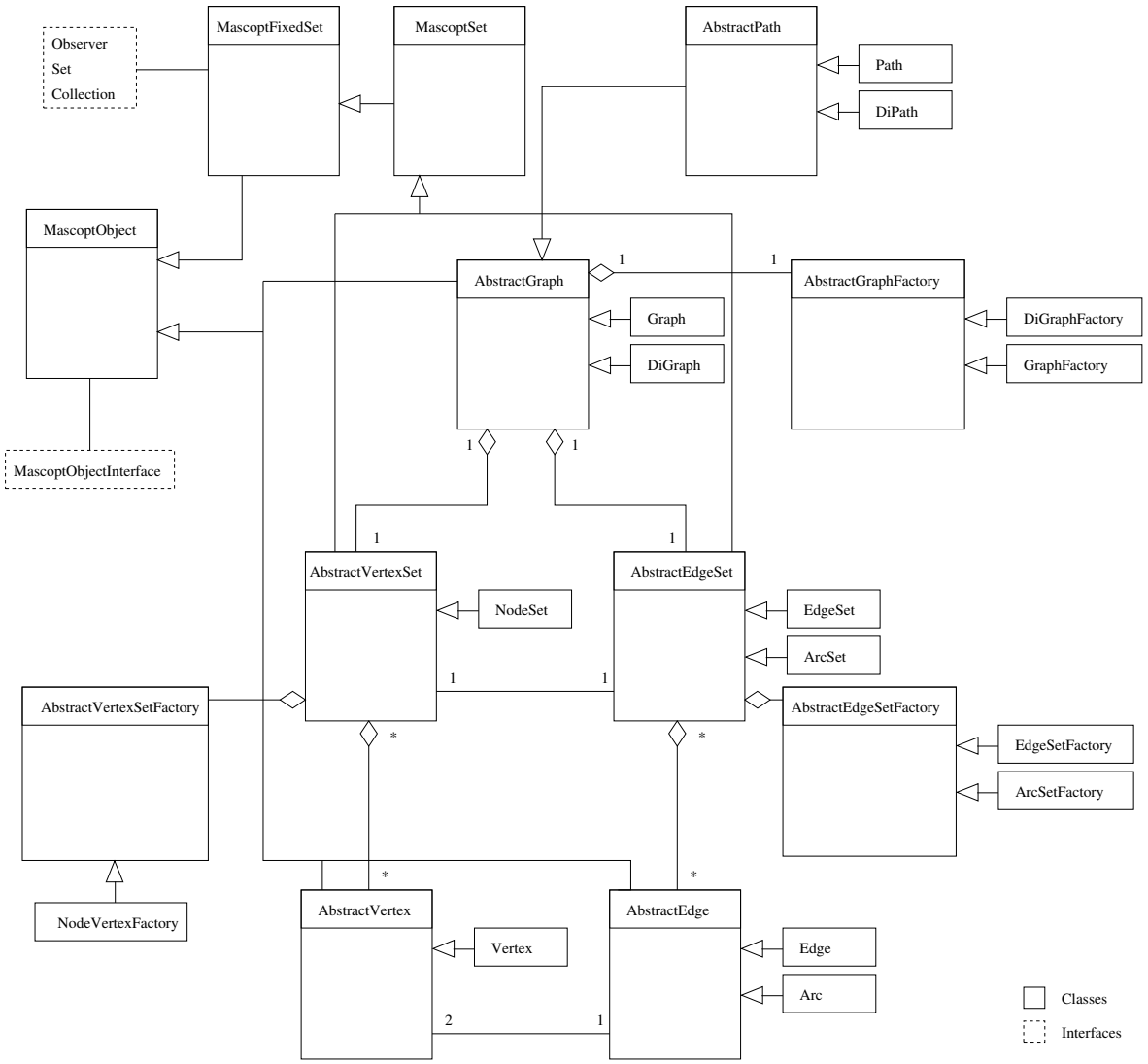


FIG. 7.2 – Diagramme de classe de la partie graphe de MASCOPT

fondamentalement de nouvelles fonctionnalités. L'objectif est de simplifier la visibilité de ces classes, au niveau de la documentation du modèle (ce qui permet d'éviter de perdre l'utilisateur potentiel dans le dédale des classes abstraites). De plus, ces classes dérivées des classes abstraites ont une fonction pédagogique dans le modèle, démontrant comment utiliser le modèle objet pour le spécialiser à l'avantage de l'utilisateur.

Nous présentons maintenant un exemple simple de la construction d'un graphe orienté dans le code source 7.2. Le code peut paraître légèrement long à écrire par rapport à d'autre langages mais nous pensons qu'il est plus naturel de commencer par construire les objets élémentaires en premier (les nœuds et les arcs), et de terminer par les objets les plus complexes, ici le graphe G_0. La représentation de ce graphe G_0 peut être trouvée en se référant à la figure 7.3 (la signification des valeurs est explicitée en section 7.4.1).

CODE SOURCE 7.2 – Création d'un graphe orienté

```
import mascoptLib.graphs.*;

public class Sample {
        public static void main (String[] args){

                Vertex n0 = new Vertex();
                Vertex n1 = new Vertex();
                Vertex n2 = new Vertex();
                Arc a0 = new Arc(n0,n2);
                Arc a1 = new Arc(n1,n2);

                VertexSet V = new VertexSet();
                V.add(n0); V.add(n1); V.add(n2);

                ArcSet E = new ArcSet(V);
                E.add(a0); E.add(a1);

                DiGraph G0 = new DiGraph(V,E);
        }
}
```

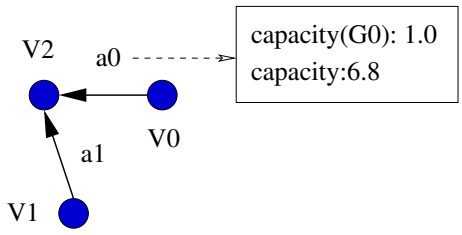


FIG. 7.3 – Un graphe G_0 valué

7.2.2. Le modèle de réseau

Notre modèle de réseau est beaucoup moins complexe que le modèle de graphe. Chaque type de réseau considéré (WDM, IP,...) possède des caractéristiques qui lui sont propres : il devient alors difficile de factoriser un modèle commun pour implémenter une classe de réseau, appelée **Network** dans MASCOPT. Cependant, un premier travail a permis de mettre en place plusieurs hypothèses pour la description d'un telle classe, mais elle reste, au moment de la rédaction de cette thèse, encore incomplète.

La classe **Network** décompose le réseau en un certain nombre de graphes. Le graphe de plus bas niveau représente la topologie physique de celui-ci. Dans le cas d'un réseau WDM, il s'agit du graphe de câbles. Le graphe de plus haut niveau est le graphe des requêtes. Ces deux graphes partagent leur ensemble de nœuds mais pas leur ensemble d'arêtes. On peut ensuite imaginer plusieurs graphes intermédiaires, suivant le type de réseau. Il peut s'agir de modéliser une topologie virtuelle définissant un nouveau support sur le support physique, de décrire des systèmes d'encapsulation de conteneur comme décrit dans la partie concernant le groupage, au chapitre 3, de décrire un flot sur le réseau physique répondant aux requêtes comme décrit au chapitre 2. Tous ces différents types de réseaux sont utilisés dans MASCOPT et le modèle générique qui les implémente utilise une classe possédant k graphes ordonnés.

Lorsqu'on utilise la classe de réseau et que l'on s'intéresse à des problèmes de télécommunications, on est très vite amené à calculer des routes pour les requêtes. S'il s'agit d'extraire un multiflot, on peut enregistrer ces données comme un ensemble de graphes de flot. S'il s'agit d'enregistrer des chemins, on préfère alors associer à chaque requête (pour nous, chaque arc du graphe de requêtes) un ensemble (du type **MascoptSet**) de chemins (du type **Path** ou **DiPath**). La classe **Network** permet alors d'ajouter ou de retirer un chemin pour une requête donnée. Par ailleurs, un système similaire permet d'enregistrer des chemins de protection, à partir d'une requête donnée et d'une panne possible.

7.3. Architecture

7.3.1. Packages

MASCOPT est constitué de huit packages JAVA™. Chaque package contient des interfaces pour orienter le développement d'un utilisateur vers le processus d'héritage des classes existantes ou directement vers la réimplémentation de classes ayant le même comportement.

- **mascoptLib.abstractGraph** : package de graphes abstraits. Il implémente la plupart des fonctionnalités de base des classes présentées dans la figure 7.2.
- **mascoptLib.algos** : package fournissant des algorithmes génériques utilisant principalement la classe abstraite **AbstractGraph** mais aussi d'autres classes du package **mascoptLib.graphs** ou **mascoptLib.networks**.
- **mascoptLib.graphs** : package pour les graphes et les graphes orientés.
- **mascoptLib.gui** : *Graphical User Interface*. Ce package permet de fournir une interface graphique, construite au dessus du package de gestion de graphes, **mascoptLib.abstractGraph**.
- **mascoptLib.io.graph** : package d'entrées/sorties. Plusieurs formats d'entrée/sortie sont fournis pour enregistrer et lire des objets du package **mascoptLib.graphs**.
- **mascoptLib.io.util** : utilitaires de manipulation de fichiers écrits par le package **mascoptLib.io.graph** package.
- **mascoptLib.networks** : package pour les réseaux.
- **mascoptLib.util** : utilitaires et algorithmes non dédiés aux graphes.

La séparation en package est fonctionnelle et les packages sont dépendants les uns des autres. Le processus de développement que nous préconisons consiste à produire des packages à l'intérieur desquelles les classes sont dépendantes les unes des autres. Ensuite, un package peut être dépendant d'un autre package mais ces dépendances doivent être clairement identifiées et minimales. Par exemple, le package de gestion de l'interface graphique dépend directement du package de gestion des graphes : l'interface graphique est une "surcouche" du package de données. Pour autant, la dépendance inverse n'existe pas.

Certains packages sont subdivisés en sous-packages, comme le package **mascoptLib.algos** qui contient des algorithmes utilisant différents types de données. Ainsi, on obtient la subdivision suivante :

- **mascoptLib.algos.abstractalgos** : package d'algorithmes travaillant sur le modèle abstrait de graphes, i.e. des classes du package **mascoptLib.abstractGraph**.
- **mascoptLib.algos.graph** : package d'algorithmes travaillant sur des graphes.
- **mascoptLib.algos.digraph** : package d'algorithmes travaillant sur des graphes orientés.
- **mascoptLib.algos.networks** : package d'algorithmes travaillant sur des réseaux.

7.3.2. Description des classes principales

Nous ne présentons ici que les classes principales relatives au modèle de donnée implémenté. Il n'est pas question de faire un liste exhaustive de toutes les classes fournies, mais de mettre en lumière la spécificité du modèle de graphes et de réseaux. Pour simplifier la lecture et pour éviter que l'énumération ne soit plus fastidieuse à lire que de raison, nous ne présentons pas les classes abstraites mais plutôt les classes du package **mascopt.graphs** dans lesquelles nous ferons référence aux classes parentes abstraites.

7.3.2.1. *Eléments simples.*

MascoptObject. Cette classe implémente le système de valuation qui permet d'enregistrer des **String**, **Double** et **Integer**. Le système de valuation est détaillé en section 7.4.1. Quasiment toutes les classes de MASCOPT implémentant un modèle de données héritent de cette classe.

Vertex. Cette classe implémente les nœuds. C'est un des éléments de base de la bibliothèque. Un **Vertex** fournit des informations à propos de ses voisins (relativement à un graphe donné), de son degré, de ses arêtes (ou arcs) entrant et sortant. Cette classe dérive de **AbstractVertex**.

Edge et Arc. Un objet **Edge** est construit à partir de deux objets du type **Vertex**. À partir d'un nœud un objet **Edge** fournit les primitives pour traverser cette arête par exemple lors d'un parcours d'un graphe. **Edge** est une classe dérivée d'**AbstractEdge**. De même, la classe **Arc** dérive de **AbstractEdge** et implémente le comportement spécifique à une arête orientée.

7.3.2.2. *Ensembles.*

MascopFixedtSet et MascoptSet. La classe **MascoptFixedSet** dérive directement de **MascoptObject**. Elle implémente le comportement d'ensemble et implémente donc les interfaces classiques d'ensemble de JAVA™, à savoir **Set** et **Collection**. L'ensemble n'étant pas ordonné, il est basé sur la classe **HashSet** permettant de retrouver un élément en un temps théorique logarithmique. **MascoptFixedSet** est un ensemble non modifiable à travers son interface publique (et donc uniquement modifiable à travers son interface protégée et dédiée au fonctionnement interne du package de graphes). La version dérivée de **MascoptFixedSet**, **MascoptSet** permet à l'utilisateur d'ajouter et retirer des éléments. De plus, des fonctionnalités propres aux ensembles complètent le système de valuation.

VertexSet. La classe **VertexSet** dérive de **AbstractVertexSet** et donc de **MascoptSet**. Elle permet de former un ensemble de nœuds (**Vertex**).

EdgeSet et ArcSet. Les classes **EdgeSet** et **ArcSet** dérivent de la classe **AbstractEdgeSet**. Elles encodent le comportement d'ensembles d'arêtes (**Edge**) et d'arcs (**Arc**).

7.3.2.3. *Graph et DiGraph.* La classe **Graph** construit un graphe non orienté à partir d'un ensemble de nœuds et d'arêtes. Elle garantit qu'à tout moment l'**EdgeSet** et le **VertexSet** utilisé sont cohérents au sens de la définition de la section 7.2.1. La classe fournit aussi des méthodes de copies et permet de construire des sous-graphes. De même, la classe **DiGraph** construit un graphe orienté à partir d'un ensemble du type **VertexSet** et d'un ensemble du type **ArcSet**. **Graph** et **DiGraph** dérivent de **AbstractGraph**.

CODE SOURCE 7.3 – Valuation du nœud v0 et de l'arc a0

```
v0.setValue("function","start");
a0.setDoubleValue("capacity", new Double(6.8));
a0.setIntegerValue("length", new Integer(110));
```

7.3.2.4. *Path et DiPath.* Un objet du type **Path** permet de construire un chemin sur un graphe. Ce chemin peut être fusionné avec un autre chemin ayant les mêmes extrémités, ce qui donne un multi-chemin. Un ensemble de primitives permet de parcourir un chemin. Il faut enfin noter que **Path** dérive de **Graph** et que l'on peut donc profiter du comportement et des fonctionnalités d'un graphe lorsque l'on manipule un chemin. Pour la classe **DiPath** le comportement est similaire mais en mode orienté : la classe dérive de **DiGraph**.

7.4. Manipulation des objets

Un des objectifs de MASCOPT est de fournir à l'utilisateur une bibliothèque facile à manipuler. Il s'agit de fournir un ensemble de primitives le plus complet possible mais aussi d'utiliser des mécanismes qui permettent de manipuler les objets du modèle de façon efficace. La liberté de manipulation donne alors une plus grande souplesse de programmation mais nécessite plus de travail pour garantir que les données restent cohérentes (plus l'accès aux données est aisé, plus il est facile de les corrompre). Nous présentons par la suite certains mécanismes de MASCOPT et les choix d'implémentation qui ont été faits.

7.4.1. Système de valuation

Comme expliqué auparavant, les objets de MASCOPT héritent tous de la classe **MascoptObject**, ceci pour factoriser le comportement commun du modèle de donnée de la bibliothèque. Le système de valuation est implémenté à ce niveau. Il permet d'enregistrer des valeurs sur un objet donné et de pouvoir relire ces valeurs ultérieurement. Les valeurs de base gérées à ce niveau sont les **String**, **Integer** et **Double**. Toutes les primitives d'accès sont fournies pour l'accès aux données ainsi que pour les types simples de JAVA™, comme les **int** et les **double**. Par exemple, l'accès à une valeur du type **Integer** se fait par un appel à **getIntegerValue()**. Chaque valeur a un nom choisi par l'utilisateur, ce qui permet d'enregistrer de multiples valeurs avec des noms différents. Aussi, des méthodes existent pour retrouver les noms utilisés (**getValueEntries()**) ou les types de données stockées (**getValueDataType()**) : elles permettent à l'utilisateur de relire des données dont il ne connaît pas le typage exact.

Nous avons choisi de restreindre les types de valeurs pour pouvoir garantir que ces données ne seront pas perdues pendant leur manipulation, que ce soit pour certains algorithmes ou pour l'écriture des données dans un fichier. Pour cette dernière, nous présentons en section 7.6.1 un exemple d'écriture fichier d'un graphe valué simple. Ces méthodes étant communes à toutes les classes du modèle de données, elles sont réutilisées pour la modélisation de certains comportements spécifiques à une classe. Par exemple, les coordonnées d'un nœud s'obtiennent par les méthodes **getX()** et **getY()** qui font un appel vers les accesseurs de valeurs.

Plus concrètement, nous montrons comment associer des valeurs aux objets précédemment construits dans le code source 7.2. Le résultat de la figure 7.3 s'obtient alors en utilisant le code source 7.3.

7.4.2. Information préconstruite en interne

L'un des objectifs de MASCOPT étant de fournir, en tant que bibliothèque, un confort d'écriture lorsque l'on écrit son propre algorithme sur des graphes ou des réseaux, MASCOPT fournit des données induites par les données instanciées par l'utilisateur. Par exemple, lorsque l'on demande l'ensemble des voisins d'un nœud u, la bibliothèque retourne un ensemble de nœuds $vs(u)$ qui a été pré-construit auparavant. Ce comportement s'oppose à une construction "à la volée" qui demande un certain temps d'exécution au moment de la demande ce qui peut poser des problèmes de performances si cet accès aux données se répète trop souvent. Néanmoins, ce temps se perd à un autre endroit du code, puisqu'il faut bien construire $vs(u)$ à un moment donné : dans MASCOPT les données d'adjacence et les ensembles d'adjacence de chaque nœud (et pour chaque graphe contenant ce nœud) sont mis à jour lorsqu'on insère un nœud ou un arc dans un ou plusieurs graphes.

La contrepartie de cette politique se paie dans l'espace mémoire occupé par les ensembles d'adjacence qui sont toujours présents en mémoire, même si l'utilisateur ne les utilise jamais, ce qui peut apparaître comme une limitation forte de la bibliothèque. Un autre effet de bord de ce choix d'implémentation peut apparaître si un algorithme exécute précisément de nombreuses fois les opérations d'insertion et de suppression d'un nœud ou d'une arête d'un graphe, provoquant une mise à jour des adjacences inutile et consommatrice de temps de calcul. Cependant, nous pensons qu'il s'agit, pour l'utilisateur, d'éviter le pire des cas et pour nous, de choisir une implémentation qui nous semble cohérente avec l'utilisation classique des données de MASCOPT. De plus, il faudra toujours choisir quel est le compromis entre le temps d'exécution des méthodes et l'espace utilisé par les données du modèle.

7.4.3. Ensembles

Etant donné le modèle de graphe choisi pour l'implémentation de MASCOPT, le concept d'ensembles est extensivement réutilisé pour stocker des données et fournir des données induites par le modèle à l'utilisateur. Ainsi, une hiérarchie de plusieurs classes a été écrite, à partir de **MascoptFixedSet** et descendant jusqu'à **EdgeSet** ou **ArcSet** par exemple. Les ensembles implémentant les interfaces d'ensemble de JAVA™, l'utilisateur peut ajouter, enlever, tester l'appartenance des objets par rapport à un ensemble. Outre la surcharge des méthodes classiques d'ensembles, certaines méthodes particulières sont implémentées, comme par exemple la modification de valeurs sur tous les éléments d'un ensemble (par exemple avec **setValueForAllElements**). La surcharge des méthodes permet, de manière classique dans la conception objet, d'effectuer des opérations spécifiques aux classes spécialisés, comme par exemple, la cohérence du type d'objet inséré dans un ensemble de nœuds ou d'arêtes.

L'introduction de **MascoptFixedSet** permet de gérer des ensembles non modifiables. Ces ensembles sont en fait modifiables par des accès protégés (*protected*) mais permettent d'éviter que l'utilisateur ne modifie les données par l'interface publique. Ce comportement est particulièrement utile pour le prêt des données internes à l'utilisateur. Par exemple, lorsque l'utilisateur demande l'adjacence d'un nœud comme décrit en section 7.4.2, l'ensemble $vs(u)$ est passé à l'utilisateur en tant qu'ensemble non modifiable. Cela permet d'éviter d'avoir à copier cet ensemble avant de le donner à l'utilisateur[7] pour éviter qu'il puisse le modifier librement. Une

[7] nous rappelons qu'en JAVA™, le passage de paramètre et le retour d'objet de fonctions se fait par "pointeur" et toujours sans copie ; la copie d'objets doit donc être explicitement demandée (et codée) ce qui implique que le don de l'ensemble d'adjacence à l'utilisateur lui permettrait de modifier un objet interne à la bibliothèque.

copie nécessiterait plus d'espace mémoire et gaspillerait du temps. Avec ce système, l'utilisateur n'a qu'un accès en "lecture seule" des données retournées. Enfin, manipuler un ensemble est beaucoup plus convivial pour l'utilisateur et permet d'accéder à plus de primitives de traitement, comparé à la manipulation d'un simple itérateur sur des données.

Les ensembles étant une des classes les plus importantes du modèle de données, nous implémentons aussi le comportement de sous-ensemble. Ce comportement amène alors de nouveaux problèmes de cohérence des données, reliés à la cohérence définie pour V et E en section 7.2.1. Nous devons garantir qu'un ensemble défini comme un sous-ensemble d'un autre le restera toujours, quel que soit les opérations qui modifient le super-ensemble. Plus concrètement, si l'utilisateur définit les ensembles **VertexSet** *VS1* et *VS2*, *VS2* étant le sous-ensemble de *VS1*, un nouveau nœud peut être ajouté à *VS2* si et seulement si ce nœud appartient déjà à *VS1*. De plus, si l'on supprime un nœud de *VS1* il sera automatiquement supprimé de *VS2*. Évidemment ce comportement est valable quel que soit le type d'ensemble choisi, par héritage de la classe mère **MascoptFixedSet**. Enfin, comme les graphes de MASCOPT sont basés sur un ensemble de nœuds et un ensemble d'arêtes, un sous-graphe hérite lui aussi de ce comportement.

7.4.4. Généricité et *design pattern factory*

Certains algorithmes n'utilisent que les caractéristiques génériques d'un graphe et peuvent même se passer de prendre en compte la particularité d'un graphe, comme par exemple le fait qu'il soit orienté ou non. Sans un mécanisme spécial, l'implémentation d'algorithmes génériques conduirait à la réécriture de ces algorithmes pour chaque type de graphe. Dans MASCOPT, ceci peut-être évité pour deux raisons principales : premièrement parce que les structures similaires héritent d'un ancêtre commun et deuxièmement parce que l'on peut utiliser le *design pattern factory* [**MDE97**], méthode de développement très connu dans le domaine de la conception objet.

Le code de l'algorithme peut être écrit dans l'ancêtre commun, ou même externalisé et dans ce cas travaillant sur cet ancêtre commun. Lorsque l'algorithme doit créer des objets de la classe dérivant de l'ancêtre commun, il demande à la classe "factory" de créer cet objet pour lui. L'algorithme ne sait donc jamais réellement quel type d'objet il manipule, si ce n'est qu'il hérite de l'ancêtre commun. Ce mécanisme permet donc très facilement d'étendre la bibliothèque MASCOPT en créant de nouvelles classes dérivées des existantes. L'utilisateur se doit alors, pour bénéficier des algorithmes existant, d'écrire la classe factory correspondant à son modèle spécialisé. Dans la structuration des packages de MASCOPT, on retrouve ces algorithmes génériques dans **mascoptLib.algos.abstractalgos**.

7.4.5. Partage des données

Comme introduit en section 7.2.1, le modèle objet de MASCOPT permet de partager des objets entre les objets composés de la bibliothèque, typiquement des nœuds entre plusieurs graphes. Ce mécanisme est particulièrement utile lorsque par exemple on construit des chemins sur un graphe : les nœuds et arêtes sont les mêmes objets pour tous les chemins. Cependant, la contrainte suivante survient : lorsque l'utilisateur demande le voisinage d'un nœud, il lui faut préciser dans quel graphe il souhaite obtenir un voisinage puisque ce nœud peut avoir différents voisinages suivant le graphe considéré.

La figure 7.4 présente deux graphes qui partagent trois nœuds et une arête (cette arête est représentée deux fois sur la figure). La figure 7.5 montre un exemple un peu plus complexe : l'ensemble de nœuds $VS1$ est partagé entre les graphes $G1$ et $G2$. Cela permet de construire deux graphes différents sans dupliquer tous les nœuds et sans dupliquer l'ensemble de nœuds

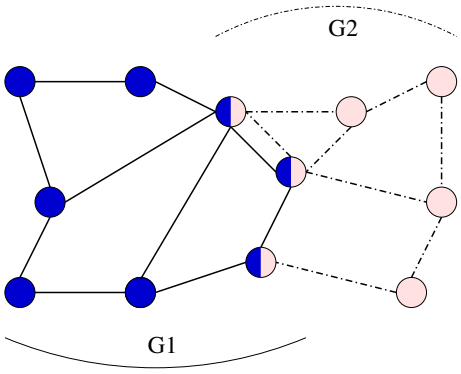


FIG. 7.4 – Partage des nœuds et des arêtes entre les graphes $G1$ et $G2$

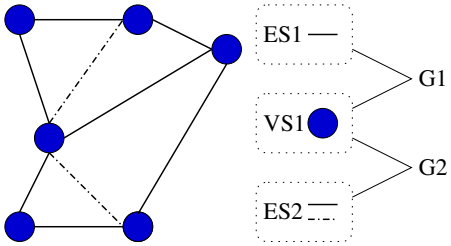


FIG. 7.5 – Partage de l'ensemble de nœuds $VS1$ avec deux ensembles d'arêtes, $ES1$ et $ES2$

CODE SOURCE 7.4 – Création du graphe orienté G3

```
ArcSet E3 = new ArcSet(V);
Arc a2 = new Arc(v0,v1);
E3.add(a0); E3.add(a2);

DiGraph G3 = new DiGraph(V,E3);
```

lui même. Les ensembles d'arêtes, eux, sont différents (sinon les deux graphes seraient identiques) : neuf arêtes sont instanciées en mémoire et sept sont communes à $ES1$ et $ES2$.

Nous montrons dans le code source 7.4 un graphe $G3$, représenté en figure 7.6, qui utilise les objets créés en section 7.2.1 par le code source 7.2.

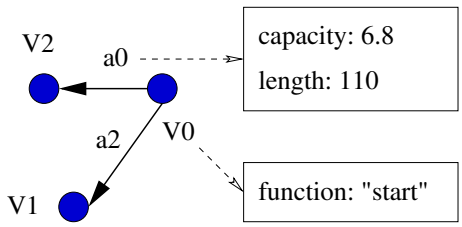


FIG. 7.6 – Un graphe orienté valué $G3$

7.4.6. La cohérence par système de messages

MASCOPT n'utilisant pas d'objet "global" ayant un pointeur sur tous les objets instanciés, les objets n'ont pas "conscience" des autres objets de plus haut niveau les utilisant. Autrement dit, un nœud ne sait pas s'il appartient à tel ou tel ensemble de nœud, et, pour prendre un autre exemple, un ensemble V ne sait pas qu'il existe un sous-ensemble V' créé sur V. Avec une telle limitation, il peut paraître difficile d'implémenter la synchronisation des ensembles et des sous-ensembles ou de garantir qu'un ensemble d'arêtes est toujours cohérent avec son ensemble de nœuds.

Nous implémentons les mécanismes qui nécessitent de modifier un objet lorsqu'un autre objet subit une modification en utilisant un système de messages. Il s'agit de lancer un message vers une liste inconnue de destinataires informant de la nature du changement. Chaque destinataire concerné effectue alors les opérations nécessaires à ce message. Les messages classiques implémentés dans MASCOPT sont les suivants :

– *add* : un objet a été inséré dans un ensemble ;
– *remove* : un objet a été supprimé d'un ensemble ;
– *value changed* : une valuation a été mise à jour.

Chaque message est constitué de son type (une **String**) ainsi que d'un pointeur vers un objet de la bibliothèque. Ainsi, si un sous-ensemble de nœuds reçoit un message du type *remove*, il peut enlever le nœud concerné (celui qui est désigné par le pointeur). Un autre exemple classique concerne l'interface graphique : lors de la mise à jour d'une coordonnée d'un nœud, l'interface reçoit un message du type *value changed* et met à jour l'objet graphique représentant ce nœud.

Ce système de messages vers une liste de destinataires est encore une fois implémenté au niveau de **MascoptObject**. Ainsi, tout objet de MASCOPT peut envoyer des messages ou en recevoir. Il suffit de coder, au niveau de chaque objet de la bibliothèque, le message à envoyer lorsqu'il subit une modification ainsi que les traitements à effectuer lors de la réception d'un message particulier.

On peut objecter à ce système un alourdissement global des performances de MASCOPT. En effet, si l'algorithme de l'utilisateur génère un grand nombre de messages dans le système (ces messages pouvant être inutiles), la vitesse d'exécution du code peut décroître significativement. Notons que ce mécanisme peut-être suspendu par l'utilisateur pour accroître les performances de son code, mais ne garantit plus la cohérence des objets. Cependant, nous pensons qu'il est plus important de garantir un comportement correct de la bibliothèques, ceci au détriment des performances, pour notamment faciliter la détection de *bugs* dans les expérimentations du programmeur.

7.5. Une courte étude de cas

Nous donnons dans cette partie une courte étude de cas qui présente l'utilisation de MASCOPT dans un problème de routage classique présenté au chapitre 2. Il s'agit de router des requêtes sur un réseau WDM modélisé par un graphe orienté $G = (V, E_G)$ où les arcs sont valués par la capacité en nombre de longueurs d'onde. Les requêtes sont stockées dans un graphe R dont l'ensemble de nœuds est partagé avec G, i.e. $R = (V, E_R)$. La sortie d'un tel problème est constitué, pour chaque requête $z \in E_R$ d'un ensemble de routes, modélisées par un objet **DiPath**. Chaque route est un chemin accroché à G c'est-à-dire un sous graphe de G. En ce

sens, il faut remarquer que, si l'utilisateur enlève un arc de $e \in E_G$ alors qu'il a créé un certain nombre de chemins, tous les chemins passant par e sont détruits car considérés comme incohérents[8].

L'implémentation de l'algorithme 3 du chapitre 2 n'est pas détaillé. Précisons seulement que, à partir du flot calculé sur G, on peut extraire, à l'aide d'une des procédures d'extraction de chemin de la section 2.2.2, les chemins un à un. Chacun de ces objets **DiPath** peuvent être fusionnés en un seul **DiPath** pour former un multi-chemin représentant le flot (pour une même requête toutes les routes ont nécessairement le même nœud source et destination). À la fin de l'algorithme, on peut enregistrer le résultat dans un fichier, chaque graphe ou chemin s'écrivant avec les valeurs de flots stockées sur les arcs : ce fichier peut alors servir d'entrée à un algorithme ultérieur.

L'utilisateur n'a donc que principalement l'algorithme à écrire, les entrées sorties étant gérées pour lui. Suivant l'algorithme, les primitives peuvent n'utiliser que le langage JAVA™ ou bien des bibliothèques externes comme CPLEX SOLVER™. En annexe C.1 nous montrons, dans le détail du code, comment interfacer un tel outil avec notre modèle de réseaux et de graphes.

7.6. Sorties fichier

La section suivante présente quelques éléments du package d'entrées/sorties de MASCOPT. Nous avons choisi d'écrire nos fichiers en XML pour favoriser l'interopérabilité avec d'éventuelles autres applications mais aussi pour permettre d'enrichir la structure XML avec de nouveaux tags, par exemple dans le cas de la spécialisation du modèle.

7.6.1. Le format MGL

Le format MGL (pour **M**ASCOPT **G**RAPH **L**IBRARY) est le format natif de MASCOPT. Ce type de fichier, assez verbeux, permet de présenter les données sous une forme humainement lisible. Le format est contrôlé par une DTD (**D***ocument* **T***ype* **D***efinition*) lors de la lecture d'un fichier. Cette DTD est présentée en annexe C.2 et donne une autre vue du diagramme de classe de la section 7.2.1.

Le code source 7.3 présente le cas du graphe orienté créé par les codes source 7.2 et 7.4 et valués par le code source 7.3.

CODE SOURCE 7.5 – MGL Fichier pour G0 et G3

```
<?xml version="1.0" ?>
<!DOCTYPE OBJECTS SYSTEM "ftp://ftp-sop.inria.fr/mascotte/mascopt/dtd/
    mgl_v1.2.dtd">

<OBJECTS>
 <VERTICES>
  <VERTEX id="V0">
   <POSITION>
     <X>50.0</X> <Y>0.0</Y>
   </POSITION>
   <VALUE type="function" dataType="String"> node0 </VALUE>
  </VERTEX>
  <VERTEX id="V1">
   <POSITION>
     <X>10.0</X> <Y>50.0</Y>
```

[8] Ils peuvent éventuellement ne pas être détruits si l'arc enlevé est au début ou à la fin du chemin : dans ce cas, le chemin est raccourci.

```
  </POSITION>
 </VERTEX>
 <VERTEX id="V2">
  <POSITION>
    <X>0.0</X> <Y>0.0</Y>
  </POSITION>
 </VERTEX>
</VERTICES>

<LINKS>
 <ARC id="AE1">
  <VERTEX_REF idref="V1"/>
  <VERTEX_REF idref="V2"/>
 </ARC>
 <ARC id="AE0">
  <VERTEX_REF idref="V0"/>
  <VERTEX_REF idref="V2"/>
  <VALUE type="Capacity" dataType="Double"> 6.8 </VALUE>
  <VALUE type="length" dataType="Integer"> 110 </VALUE>
 </ARC>
 <ARC id="AE2">
  <VERTEX_REF idref="V0"/>
  <VERTEX_REF idref="V1"/>
 </ARC>
</LINKS>

<SETS>
 <VERTEX_SET id="VS0">
  <VERTEX_REF idref="V0"/>
  <VERTEX_REF idref="V1"/>
  <VERTEX_REF idref="V2"/>
 </VERTEX_SET>
 <ARC_SET id="AES0">
  <ARC_REF idref="AE0"/>
  <ARC_REF idref="AE1"/>
 </ARC_SET>
 <ARC_SET id="AES3">
  <ARC_REF idref="AE0"/>
  <ARC_REF idref="AE2"/>
 </ARC_SET>
</SETS>

<GRAPHS>
 <DIGRAPH id="G0">
  <VERTEX_SET_REF idref="VS0"/>
  <ARC_SET_REF idref="AES0"/>
 </DIGRAPH>
 <DIGRAPH id="G3">
  <VERTEX_SET_REF idref="VS0"/>
  <ARC_SET_REF idref="AES3"/>
 </DIGRAPH>
</GRAPHS>
</OBJECTS>
```

7.6.2. La vision DOM du XML

Le processus d'écriture est construit autour d'un analyseur de syntaxe ou *parser* XML, technologie nécessaire à la bonne manipulation de données XML. Nous utilisons XERCES pour

réaliser la lecture ou l'écriture des données, qui sont des classes issues du projet APACHE. XERCES propose deux parsers différents : SAX qui manipule les données XML par événements et DOM qui manipule les données sous la forme d'un arbre. En fait, DOM utilise le parseur SAX puisqu'il est forcément nécessaire de lire le fichier séquentiellement (ce qui génère les événements de SAX du type "entrée dans un tag", "sortie d'un tag", ...).

Une première version du package d'entrée sortie a été écrit à l'aide de SAX. Cependant, SAX ne répondait que partiellement au besoin, car il devenait difficile de proposer à l'utilisateur une manière simple pour lui permettre d'écrire ses propres classes. L'écriture utilisant SAX est basée sur la classe **MGLWriter** qui reçoit les objets de MASCOPT, devant être écrits dans un fichier. La classe **MGLWriter** s'occupe de filtrer les types des objets pour identifier les sous-objets qui composent l'objet à écrire (par exemple, identifier l'ensemble de nœuds d'un graphe, puis les nœuds de l'ensemble de nœuds). Ces objets étant identifiés, ils sont mis dans la pile des objets à écrire. À la fin, on écrit sequentiellement tous les objets à l'aide de SAX. Avec une telle modélisation, il est alors impossible pour l'utilisateur de donner à **MGLWriter** un nouveau type d'objet à écrire. Il faudrait en fait créer une classe héritant de **MGLWriter** qui ajouterait le comportement voulu, ce qui devient complexe et ne nous a pas permis, après de nombreux essais, de trouver une manière élégante (et robuste) pour l'écrire.

Une deuxième version de **MGLWriter** a permis de résoudre ce problème en utilisant DOM. DOM est basé sur un arbre représentant la structure XML en modélisant chaque profondeur par un nœud. Dans un tel arbre, les feuilles de l'arbre sont les tags du document XML de la plus grande profondeur. Il s'agit alors de créer les nœuds DOM en fonction des objets à écrire. Pour les objets composés, il suffit de permettre un appel récursif pour la création des nœuds DOM des sous-objets. On déplace alors le processus d'écriture à l'intérieur des objets de MASCOPT, ce qui diffère totalement de la politique présente qui externalisait l'écriture dans **MGLWriter**.

Ainsi, chaque objet est capable de "s'écrire" dans un arbre DOM qu'on lui donne en paramètre. Lorsque par exemple, on demande à un graphe $G = (V, E)$ de s'écrire, il exécute une partie de son code qui crée le nœud DOM correspondant. Ensuite, le graphe appelle la méthode d'écriture de V et de E. À leur tour, V et E s'écrivent dans l'arbre DOM et appellent l'écriture respectivement de leurs nœuds et arêtes. Avec un tel processus, il peut facilement arriver que la méthode d'écriture d'un nœud donné soit appelée plusieurs fois, par exemple si le nœud est utilisé par deux graphes et que ces deux graphes sont écrits dans un même fichier. Pour éviter des écritures multiples, un objet cherche au travers de l'arbre DOM le tag qui lui correspond : s'il ne le trouve pas alors il s'écrit, sinon il considère qu'il a déjà été écrit auparavant. Cette recherche arborescente est une fonctionnalité exclusive à DOM et nécessite d'avoir un modèle de donnée de l'arbre en mémoire, avant le processus de sérialisation.

Ainsi, si l'utilisateur souhaite ajouter des objets ou spécialiser des objets de la bibliothèque, il lui suffit d'implémenter la méthode d'écriture dans un arbre DOM en prenant garde au problème de la multiple écriture. En ce qui concerne la lecture, le processus est inverse : une classe **MGLReader** construit l'arbre DOM correspondant au fichier et se charge ensuite de construire les objets de MASCOPT dans l'ordre souhaité (les nœuds avant les ensembles de nœuds par exemple). Pour des classes de l'utilisateur, il suffit de spécialiser **MGLReader** pour décrire la correspondance entre un tag donné et l'objet à construire (en précisant éventuellement les ordres nécessaires).

7.6.3. Comment étendre le format : l'exemple du MGX

Nos efforts pour l'élaboration d'un système de lecture/écriture extensible permettent donc d'écrire des objets spécialisés de la bibliothèque ou même de nouveaux objets. Nous revenons

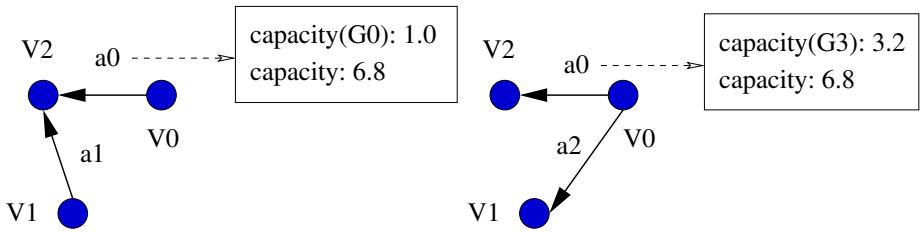


FIG. 7.7 – $G0$ et $G3$ avec des valeurs avec contexte

sur le format d'écriture MGL et nous montrons comment il peut être étendu dans un cas concret pour définir le format dérivé MGX (pour **M**ascopt **G**raph e**X**tended).

Lors de l'écriture des valeurs sur un objet de MASCOPT, on peut vouloir enregistrer plusieurs valeurs différentes sur un même objet suivant qu'il est utilisé dans tel ou tel graphe. Pour ce problème, nous adaptons le système de valuation présenté en section 7.4.1 pour introduire la notion de contexte lors de l'enregistrement d'une valeur. De même, lors de la lecture, l'utilisateur doit préciser le contexte dans lequel il lit une valeur.

Dans la section 7.4.2, l'arc $a0$ est valué par $Capacity = 6.8$ et cette valeur est la même dans tous les graphes contenant cet arc. Avec les deux graphes $G0$ et $G3$ et à l'aide de la notion de contexte, nous montrons comment créer des valuations différentes comme dans le code source 7.6. Au niveau des fichiers, la différence entre le fichier MGL et la version MGX est détaillée dans le listing 7.7. Il faut aussi noter que la valeur sans contexte 6.8 devient la valeur par défaut : sans la précision d'un contexte, le système de valuation retourne cette valeur. La figure 7.7 présente les deux graphes résultants.

CODE SOURCE 7.6 – Valuation de l'arc $a0$ avec des contextes

```
a0.setDoubleValue("Capacity",G0, new Double(1.0));
a0.setDoubleValue("Capacity",G3, new Double(3.2));
```

CODE SOURCE 7.7 – Partie modifiée du code source 7.5

```
<ARC id="AE0">
 <VERTEX_REF idref="V0"/>
 <VERTEX_REF idref="V2"/>
 <VALUE type="Capacity" dataType="Double"> 6.8 </VALUE>
 <VALUE type="Capacity" dataType="Double" context="G0"> 1.0 </VALUE>
 <VALUE type="Capacity" dataType="Double" context="G3"> 3.2 </VALUE>
</ARC>
```

7.7. L'interface graphique

Une interface graphique a été développée pour faciliter la visualisation des résultats de nos expérimentations. Cette interface graphique est une surcouche indépendante du modèle de donnée. Elle fonctionne par la réception de messages, comme explicité dans la section 7.4.6. Ainsi, pour tous les événements du modèle de donnée, tel l'ajout d'un nœud à un graphe, le changement d'une valeur, l'interface graphique reçoit une notification et exécute les procédures de mise à jour des objets graphiques.

À partir du package graphique, nous avons développé deux applications graphiques génériques : un éditeur et un visualiseur de graphes, que nous présentons brièvement.

L'éditeur. L'éditeur permet de modifier un graphe par l'intermédiaire d'une interface graphique. L'accès aux primitives **add**, **remove** ou **setValue** est recouvert par un code capturant les événements graphiques.

Le visualiseur. Le visualiseur est une application complète qui permet d'afficher un nombre important de graphes. Il est à la fois multi-couches et multi-vues. L'utilisateur peut donc créer plusieurs vues (pour par exemple visualiser plusieurs parties d'un même graphe), chaque vue étant constituée de plusieurs couches ou "layers". L'introduction de la notion de layer permet de grouper plusieurs graphes dans une même couche : ces graphes sont empilés, ce qui permet de les dessiner en superposition. Des primitives graphiques permettent alors de masquer ou d'afficher un graphe précis, ou une couche entière (c'est-à-dire un groupe de graphes) ou une vue totale. Des labels peuvent être créés sur les nœuds ou les arêtes/arcs graphiques, pour l'affichage des valeurs enregistrés sur les objets du modèle de données.

Par ailleurs, le visualiseur n'est pas seulement une application dont l'accès est purement graphique. Il est assez modulaire pour être piloté à travers un code source JAVA™, permettant à l'utilisateur de créer/détruire des vues et couches dans son propre code. C'est la classe **MascoptViewer** qui masque la complexité du code graphique et qui sert de point central de pilotage de l'interface lorsqu'on l'utilise par le code.

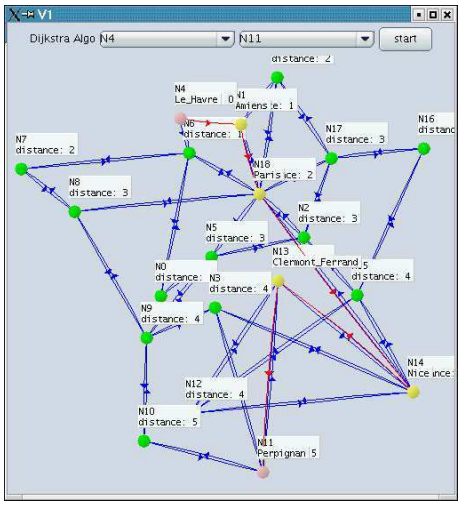


FIG. 7.8 – Le réseau R_1 vu dans le visualiseur

7.8. Perspectives

Notre travail, dans MASCOPT, est divisé en différents "workpackages" récapitulés en table 7.1. La seconde distribution publique (version 1.2) contient le workpackage WP1 et une partie significative de WP7 et WP8. Au moment de l'écriture de cette thèse, nos efforts se portent sur le package WP2 qui s'appuie sur les développements du package WP1. En particulier, nous souhaitons développer et implémenter des modèles de réseaux plus complets pour formaliser les entrées/sorties de nos algorithmes dédiés aux réseaux.

Par rapport aux développements menés dans l'équipe MASCOTTE, la bibliothèque MASCOPT est un projet à long terme, à l'inverse du projet RNRT PORTO, limité à une durée de trois ans. Ainsi, les développements qui ont eu lieu dans le cadre de cette thèse ont toujours

WPX	Workpackage	Description
WP1	Graphes	Modèle et manipulation de graphes
WP2	Réseaux	Modèle et manipulation de réseaux
WP3	Réseaux virtuels	Interaction entre différents types de réseaux
WP4	Programmation linéaire	Optimisation linéaire dans les réseaux
WP6	Données expérimentales	Données réalistes de réseaux
WP7	Interface graphique	Représentation de graphes et de réseaux
WP8	Entrées/Sorties	Modélisation XML, formats d'échange
WP9	Algorithmes sur les graphes et graphes orientés	Algorithmes de la théorie des graphes

TAB. 7.1 – Plan de travail de MASCOPT

cherché des solutions robustes et fiables et qui permettront, dans les développements futurs, de poursuivre cet effort.

Conclusion et perspectives

Cette deuxième partie nous a permis de présenter deux réalisations logicielles : PORTO dont le développement est maintenant clos, et MASCOPT dont le développement se poursuit activement.

Le logiciel PORTO a permis de valider certaines approches algorithmiques pour la conception de réseaux optiques et notamment dans le cas des réseaux à trois niveaux hiérarchiques. L'implémentation de ces algorithmes a démontré la nécessité d'heuristiques ou d'approximation pour réussir à répondre aux instances réelles de nos réseaux.

L'architecture de développement choisie a permis d'implémenter des algorithmes de routage, de groupage et de protection en permettant à d'éventuels développeurs de continuer à intégrer de nouveaux algorithmes dans le système de plugin. Au niveau de l'interface utilisateur, notre souci a été de présenter de manière claire les résultats obtenus au niveau des câbles optiques et des équipements des nœuds, en mettant l'accent sur les détails du brassage des longueurs d'onde, des bandes et des fibres. L'interface permet alors à l'opérateur de réaliser une analyse fine de l'architecture de son réseau pour éventuellement réaliser certaines opérations manuellement.

La réalisation de PORTO a mis en évidence les limites du processus de développement lorsqu'un logiciel est spécifique à un modèle précis de données. Pour obtenir l'efficacité souhaitée au niveau des algorithmes développés, le modèle de donnée est dédié à la représentation des réseaux optiques et ne permet plus une réutilisation efficace des algorithmes ou du modèle pour des problèmes connexes.

Dans ce contexte, nous avons présenté la bibliothèque MASCOPT et les objectifs de développement particulier de cette bibliothèque. Nous avons détaillé les particularités du modèle de graphes et de réseaux ainsi que les choix d'implémentation qui ont été faits. Des comportements particuliers pour la gestion des ensembles et sous-ensembles, la généricité du modèle, la valuation des objets caractérisent la spécificité de nos développements. L'implémentation du modèle objet correspondant nous a conduit à créer un format de données dédié, utilisant les avantages du formalisme XML.

La suite de nos efforts dans MASCOPT portera principalement sur l'enrichissement du modèle de réseaux et la poursuite de l'implémentation d'algorithmes d'optimisation dans les graphes et les réseaux. Bien entendu, MASCOPT constituera la plate-forme privilégiée pour l'expérimentation des résultats théoriques algorithmiques à venir. Nous souhaitons aussi que MASCOPT puissent être utilisé au sein des communautés algorithmiques et télécoms (projet IST CRESCCO, communauté TAROT, ...) et puisse faciliter les échanges et collaborations (CEMAGREF, université de Fortaleza) au niveau de la partie expérimentale de nos recherches.

Bilan

Cette thèse est découpée en deux parties, l'une concernant les aspects algorithmiques et l'autre les aspects logiciels. Nous ne rappellerons pas les éléments des conclusions déjà inclus dans chacune des parties : nous donnons simplement un bilan au niveau de la réussite des objectifs fixés soit au début soit en cours de cette thèse.

Pour la partie algorithmique, nous avons pu vérifier qu'il faut souvent faire appel à des heuristiques dès lors que l'on souhaite traiter des instances non triviales de problèmes. La validation des algorithmes que nous avons proposé passe par l'expérimentation sur des instances de réseaux réels. Tout ceci implique alors un travail supplémentaire non négligeable en plus des autres aspects de la démarche comme par exemple la modélisation par la programmation linéaire ou l'énoncé de la complexité des problèmes.

Ainsi, c'est par souci d'efficacité et avec pour objectif de pouvoir tester le plus d'algorithmes possibles dans un temps imparti que nous avons proposé la bibliothèque MASCOPT, qui est la contribution principale de la seconde partie. À ce jour, nous pensons avoir fait le bon choix (développer un outil générique) bien que nous n'ayons pas encore eu l'occasion de développer autant d'expériences que souhaité et que MASCOPT soit encore en cours d'amélioration. La diffusion de cette bibliothèque conjointement à la publication des algorithmes proposés dans cette thèse devrait permettre très prochainement à tout un chacun de vérifier les résultats énoncés, du moins pour les derniers travaux effectués avec MASCOPT, par exemple pour nos algorithmes de protection.

Bibliographie

AAG+04. S. Alouf, E. Altman, J. Galtier, J.-F. Lalande, and C. Touati. Un algorithme d'allocation de bande passante satellitaire. Technical Report RR-5172, INRIA Sophia Antipolis, 2004 route des lucioles - BP 93 - FR-06902 Sophia Antipolis, April 2004.

AAG+05a. S. Alouf, E. Altman, J. Galtier, J.-F. Lalande, and C. Touati. Quasi-optimal bandwidth allocation for multi-spot mftdma satellites. In *IEEE Infocom*, Miami, USA, March 2005. To appear.

AAG+05b. S. Alouf, E. Altman, J. Galtier, J.-F. Lalande, and C. Touati. Quasi-optimal resource allocation in multi-spot MFTDMA satellite networks. In M. Cheng, Y. Li, and D.-Z. Du, editors, *Combinatorial Optimization in Communication Networks*. Kluwer Academic Publishers, 2005. To appear.

ACB97. J. Armitage, O. Crochat, and J.-Y. Le Boudec. Design of a survivable WDM photonic network. In *IEEE Infocom*, pages 244–252, April 1997.

AGT02. E. Altman, J. Galtier, and C. Touati. A survey on TDMA satellite systems and slot allocation, June 2002. `http://www-sop.inria.fr/maestro/personnel/Eitan.Altman/PAPERS/tdmasurve%y.pdf`.

alp. IBM alphaWorks. Graph foundation classes for Java. `http://www.alphaworks.ibm.com/tech/gfc`.

AN01. J. Akkanen and J. K. Nurminen. Case study of the evolution of routing algorithms in a network planning tool. *Journal of Systems and Software*, 58(3) :181–198, 2001.

AV79. D. Angluin and L.G. Valiant. Fast probabilistic algorithms for hamilton circuits and matchings. *Journal of Computer and System Science*, 18 :155–193, 1979.

Ba99. P. Batchelor and al. Ultra high capacity optical transmission networks : Final report of action COST 239. Technical Report ISBN 953-184-013-X, Faculty of Electrical Engineering and Computing, HR, Zagreb, 1999.

BB97. S. Baroni and P. Bayvel. Wavelength requirements in arbitrarily connected wavelength-routed optical networks. *IEEE/OSA Journal of Lightwave Technology*, 1997.

BBG99. S. Baroni, P. Bayvel, and R. J. Gibbens. Analysis and design of resilient multifiber wavelength-routed optical transport networks. *Journal of lightwave technology*, 17(5) :743–758, 1999.

BC03. J.-C. Bermond and D. Coudert. Traffic grooming in unidirectional WDM ring networks using design theory. In *ICC 2003, IEEE International Conference on Communications*, 11-15 May 2003.

BCCT01. J-C. Bermond, L. Chacon, D. Coudert, and F. Tillerot. Cycle Covering. In *International Colloquium on Structural Information and Communication Complexity – SIROCCO*, pages 21–34, 27-29 June 2001.

BCJ+97. Matthias Berger, Michel Chbat, Amaury Jourdan, Michel Sotom, Piet Demeester, Bart Van Caenegem, Poul Gødsvang, Bernard Hein, Manfred Huber, Reinhard März, Alain Leclert, Torodd Olsen, Gerald Tobolka, and Tom Van den Broeck. Pan-european optical networking using wavelength division multiplexing. *IEEE Communications Magazine*, 35(4) :82–88, April 1997.

BCL+03. M. Bouklit, D. Coudert, J.-F. Lalande, C. Paul, and H. Rivano. Approximate multicommodity flow for WDM networks design. In J. Sibeyn, editor, *International Colloquium on Structural Information and Communication Complexity –SIROCCO*, number 17 in Proceedings in Informatics, pages 43–56, Umea, Sweden, 2003. Carleton Scientific.

BCLR03. M. Bouklit, D. Coudert, J.-F. Lalande, and H. Rivano. Approximation combinatoire de multiflot fractionnaire : améliorations. In INRIA, editor, *AlgoTel*, Banyuls-surmer, France, mai 2003.

BCM03. J.-C. Bermond, D. Coudert, and X. Munoz. Traffic grooming in unidirectional wdm ring networks : the all-to-all unitary case. In *IFIP Working Conference on Optical Network Design and Modelling*, pages 1135–1153, 3-5 February 2003.

BHP98. B. Beauquier, P. Hell, and S. Perennes. Optimal wavelength-routed multicasting. *Discrete Applied Mathematics*, 84 :15–20, 1998.

BK95. A. Birman and A. Kershenbaum. Routing and wavelength assignment methods in single-hop all-optical networks with blocking. In *IEEE Infocom*, pages 431–438, New-York, USA, 1995.

BPS02. B. Beauquier, S. Pérennes, and M. Syska. Efficient access to optical bandwidth, routing and grooming in WDM networks : State-of-the-art survey. IST CRESCCO report, Projet MASCOTTE (CNRS/INRIA/UNSA), Sophia Antipolis, July 2002.

BS97. R.A. Barry and S. Subramaniam. The max-sum wavelength assignment algorithm for WDM ring networks. *IEEE Optical Fiber Communication*, pages 121–122, February 1997.

BW00. C. Baworntummarat and L. Wuttisittikulkij. On the comparison of optical WDM mesh network protection strategies. In *IEEE Military Communications Conference*, number 1, pages 886–891, October 2000.

CAQ04. X. Cao, V. Anand, and C. Qiao. Multi-layer versus single-layer optical cross-connect architectures for waveband switching. In *IEEE Infocom*, Hong Kong, China, March 2004.

CB98. O. Crochat and J. Boudec. Design protection for WDM optical networks. *IEEE Journal on Selected Areas in Communications*, 16(7) :1158–1165, 1998.

CCMV04. F. Chauvet, P. Chrétienne, P. Mahey, and B. Vatinlen. Minimisation du nombre de chemins décomposant un flot. In INRIA, editor, *AlgoTel*, pages 39–43, Batz-surmer, France, mai 2004.

Che52. H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of mathematical statistics*, 23, 1952.

Chv83. V. Chvatal. *Linear programming*. W. H. Freemann and Company, 1983.

CM00. A. L. Chiu and E. H. Modiano. Traffic grooming algorithms for reducing electronic multiplexing costs in WDM ring networks. *IEEE/OSA Journal of Lightwave Technology*, 18(1) :2–12, January 2000.

CR02. D. Coudert and H. Rivano. Lightpath assignment for multifibers WDM optical networks with wavelength translators. In *IEEE Globecom*, Taiwan, November 2002. cdrom, OPNT-01-5.

CRD04. B. Chen, G. N. Rouskas, and R. Dutta. Traffic grooming in WDM ring networks with the min-max objective. In *Networking*, pages 174–185, Athens, Greece, May 2004. LNCS 3042.

CSC02. H. Choi, S. Subramaniam, and H. Choi. On double-link failure recovery in WDM optical networks. In *IEEE Infocom*, pages 808–816, New-York, USA, June 2002.

DR02. R. Dutta and G. N. Rouskas. Traffic grooming in WDM networks : Past and future. *IEEE Network*, 16(6) :46–56, 2002.

DSS03. P. Datta, M. Sridharan, and A. K. Somani. A simulated annealing approach for topology planning and evolution of mesh-restorable optical networks. In *IFIP Working Conference on Optical Networks Design and Modelling*, pages 23–40, Budapest, Hungary, 2003.

DW60. G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8 :101–111, 1960.

EE99. T. ElBatt and A. Ephremides. Frequency reuse impact on the optimum channel partitioning for hybrid wireless systems. In *International Mobile Satellite Communications Conference*, Ottawa, Canada, June 1999.

EHS00. G. Ellinas, A. Gebreyesus H., and T. E. Stern. Protection cycles in mesh WDM networks. *IEEE Journal on Selected Areas in Communications*, 18(10) :152–156, October 2000.

FF62. L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.

FHW80. S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10 :111–121, 1980.

Fle00. L. Fleischer. Approximating fractional multicommodity flows independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4) :505–520, 2000.

FMSN00. D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, 34(2) :251–281, February 2000.

FPRB. M. Forster, A. Pick, M. Raitner, and C. Bachmaier. GTL : Graph template library. `http://infosun.fmi.uni-passau.de/GTL`.

FS03. M. T. Frederick and A. K. Somani. A single-fault recovery strategy for optical networks using subgraph routing. In *IFIP Working Conference on Optical Network Design and Modeling*, volume 1, pages 549–568, 2003.

FV00. A Fumagalli and L. Valcarenghi. IP restoration vs. WDM protection : Is there an optimal choice ? *IEEE Network*, 14(6) :34–41, November 2000.

GCW82. I. S. Gopal, D. Coppersmith, , and C. K. Wong. Minimizing packet waiting time in a multibeam satellite system. *IEEE Transactions on Communications*, 30(2) :305–316, February 1982.

GJ79. Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP- Completeness*. W.H. Freeman, 1979.

GK98. N. Garg and J. Konemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.

GK03. H. F. Geerdes and H. Karl. The potential of relaying in cellular networks. In *International Network Optimization Conference*, pages 237–242, Evry/Paris, France, October 2003.

GLS81. M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1 :169–197, 1981.

Gmb. Algorithmic Solutions Software GmbH. LEDA. `http://www.algorithmic-solutions.com/enleda.htm`.

GR00. J. Gruber and R. Ramaswami. Moving toward all-optical networks. *Lightwave Magazine*, pages 60–68, December 2000.

GRS00. O. Gerstel, R. Ramaswami, and G.H. Sasaki. Cost-effective traffic grooming in WDM rings. *IEEE/ACM Transactions on Networking*, 8(5) :618–630, October 2000.

GRW00. O. Gerstel, R. Ramaswami, and W. Wang. Making use of a two-stage multiplexing scheme in a WDM network. In *OSA/SPIE Optical Fiber Communication Conference and Exposition*, volume 3, pages 44–46, 2000.

GT03. P. Ghobril and S. Tohmé. Wavelength rearrangement. In *IFIP Working Conference on Optical Network Design and Modeling*, volume 2, pages 939–951, 2003.

GW85. I. S. Gopal and C. K. Wong. Minimizing the number of switchings in an SS/TDMA system. *IEEE Transactions on Communications*, 33(6) :497–501, June 1985.

Hil96. D. R. C. Hill. *Object-Oriented Analysis and Simulation*. Addison-Wesley Longman, 1996.

HK99. H. Harai and F. Kubota. Dimensioning of MP/WDM networks with failure restoration. In *IEEE Globecom*, pages 1042–1047, Rio de Janeiro, December 1999.

HO01. S. Hochbaum and E. Olinick. The bounded cycle cover problem. *Informs Journal on Computing*, 13(2) :104–119, 2001.

HPS02. G. Huiban, S. Pérennes, and M. Syska. Traffic grooming in WDM networks with multi-layer switches. In *IEEE International Conference on Communications*, pages 2896–2901, New-York. USA, April 2002. Cdrom.

HS96. J. C. Hardwick and J. Sipelstein. Java as an intermediate language. Technical Report CMU-CS-96-161, School of Computer Science, Carnegie Mellon University, August 1996.

HSKO99. K. Harada, K. Shimizu, T. Kudou, and T. Ozeki. Hierarchical optical path cross-connect systems for large scale WDM networks. In *IEEE Optical Fiber Communication*, pages 356–358, San Diego, USA, 1999.

HV98. E. Hyytiä and J. Virtamo. Wavelength assignment and routing in WDM networks. In *Fourteenth Nordic Teletraffic Seminar, NTS-14*, Copenhagen, Denmark, August 1998.

IMG98. R.R. Iraschko, M.H. MacGregor, and W.D. Grover. Optimal capacity placement for path restoration in STM or ATM mesh-survivable networks. *IEEE / ACM Transactions on Networking*, 6(3) :325–336, June 1998.

IMR01. Alcatel Research & Innovation, Projet MASCOTTE(CNRS/INRIA/UNSA), and France Télécom R&D. Planification et optimisation des réseaux de transport optiques. Technical report, Rapport final RNRT PORTO, Sophia Antipolis, Décembre 2001.

Jar02. A. Jarry. Disjoint paths in symmetric digraphs. In *International Colloquium on Structural Information and Communication Complexity – SIROCCO*, pages 211–222, Andros, Greece, June 2002. Carleton.

Kar80. I. A. Karapetyan. On coloring of arc graphs. *Doklady Akad. Nauk Armianskoi CCP*, 70(5) :306–311, 1980. In Russian.

KC98. V. Kann and P. Crescenzi. Maximum independent set, 1998. `http://www.nada.kth.se/~viggo/wwwcompendium/wwwcompendium.html`.

Kum98. V. Kumar. Approximating circular arc coloring and bandwidth allocation in all-optical ring networks. In *Proc. of the International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'98)*, volume 1444 of *Lecture Notes in Computer Science*, pages 147–158. Springer-Verlag, 1998.

LD02. B. Liau and B. Decocq. Réseaux optiques du futur : optimisation des réseaux. In *Actes des Conférences France Télécom Recherche*, number 19, Juin 2002.

LE01. A. Lardiés and G. Ester. Optimal network design of ultra-long-haul transmission networks. Alcatel Telecommunications Review, 2^{nd} Quarter 2001.

LPS03. J.-F. Lalande, S. Pérennes, and M. Syska. Groupage dans les réseaux dorsaux WDM. In Université d'Avignon et des Pays de Vaucluse, editor, *ROADEF*, number 5 in Proceedings in Informatics, pages 254–255, Avignon, France, 2003.

LSV04. J.-F. Lalande, M. Syska, and Y. Verhoeven. Mascopt - a network optimization library : Graph manipulation. Technical Report RT-0293, INRIA Sophia Antipolis, 2004 route des lucioles - BP 93 - FR-06902 Sophia Antipolis, April 2004.

LSV05. J.-F. Lalande, , M. Syska, and Y. Verhoeven. Arrondi aléatoire et protection des réseaux WDM. In *ROADEF*, 2005. To appear.

LYK+02. M. Lee, J. Yu, Y. Kim, C-H. Kang, and J. Park. Design of hierarchical crossconnect WDM networks employing a two-stage multiplexing scheme of waveband and wavelength. *IEEE Journal on Selected Areas in Communications*, 20(1) :166–171, January 2002.

MBR96. B. Mukherjee, D. Banerjee, and S. Ramamurthy. Some principles for designing a wide area WDM optical network. *IEEE/ACM Transactions on Networking*, 4(5) :684–696, 1996.

MDE97. T. D. Meijler, S. Demeyer, and R. Engel. Making design patterns explicit in FACE - A framework adaptive composition environment. In M. Jazayeri and H. Schauer, editors, *Software Engineering Conference*, pages 94–110. Springer–Verlag, 1997.

MIB+04. A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot. Characterization of failures in an IP backbone. In *IEEE Infocom*, Hong Kong, China, March 2004.

MM01. G. Mohan and C. Siva Ram Murthy. Routing dependable connections in WDM optical. *IEEE Computer Communications*, 24 :1225–1241, 2001.

MM04. H. V. Madhyastha and C. Siva Ram Murthy. Efficient dynamic traffic grooming in service-differentiated wdm mesh networks. *Journal of Computer and Telecommunications Networking*, 42(2) :221–235, 2004.

MNU97. K. Mehlhorn, S. Naher, and C. Uhrig. The LEDA platform of combinatorial and geometric computing. In *International Colloquium on Automata, Languages and Programming*, pages 7–16, 1997.

Mos04. C. Mosse. Implémentation d'algorithmes pour les réseaux optiques WDM. Rapport de fin d'études d'Elève Ingénieur Maître, IUP GMI d'Avignon, 2004.

NW88. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

OMV97. A. Ouorou, P. Mahey, and J.-P. Vial. A survey of algorithms for convex multicommodity flow problems. Technical Report 97.3, Department of Management Studies, University of Geneva, Switzerland, 1997.

Pre00. L. Prechelt. An empirical comparison of seven programming languages. *IEEE Computer*, 33(10) :23–29, October 2000.

Rag94. P. Raghavan. Randomized approximation algorithms in combinatorial optimization. Technical report, IBM Research Division, 1994.

Riv03. H. Rivano. *Algorithmique et télécommunications : Coloration et multiflot approchés et applications aux réseaux d'infrastructure*. PhD thesis, Université de Nice-Sophia Antipolis, novembre 2003.

RS95. N. Robertson and P. D. Seymour. Graph minors. xiii : the disjoint paths problem. *Journal of Combinatorial Theory Series B*, 63(1) :65–110, January 1995.

RT87. P. Raghavan and C. D. Thompson. Randomized rounding : A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 1987.

Sal. J. M. Salvo. Openjgraph - Java graph and graph drawing project. `http://openjgraph.sourceforge.net`.

SB99. T.E. Stern and K. Bala, editors. *Multiwavelength Optical Networks : A Layered Approach*. Addison-Wesley, 1999.

Sch94. A. Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM Journal on Computing*, 23(4) :780–788, August 1994.

SP03. D. A. Schupke and R. G. Prinz. Capacity efficiency and restorability of path protection and rerouting in WDM networks subject to dual failures. *Photonic Network Communications*, September 2003.

SS99. A. A. M. Saleh and J. M. Simmons. Architectural principles of optical regional and metropolitan access networks. *IEEE/OSA Journal of Lightwave Technology*, 17(12) :2431–2448, December 1999.

SS02. M.D. Swaminathan and K.N. Sivarajan. Practical routing and wavelength assignment algorithms for all optical networks with limited wavelength conversion. In *IEEE International Conference on Communications*, volume 25, pages 2750–2755, New-York, April 2002.

SSS02. M. Sridharan, M. V. Salapaka, and A. Somani. A practical approach to operating survivable wdm networks. *IEEE Journal on Selected Areas in Communications on WDM-based Network Architectures*, 20(1), January 2002. Special Issue on WDM Based Network Architectures.

Ta03. R. Tamassia and al. An overview of JDSL 2.0, the data structures libary in Java. Technical report, 2003.

Tof95. B. Toft. Coloring, stable sets and perfect graphs. In Graham, Grötschel, and Lovász, editors, *Handbook of combinatorics*, volume 1, chapter 4, pages 233–288. North Holland, 1995.

Tuc75. A. Tucker. Coloring a family of circular arcs. *SIAM Journal of Applied Mathematics*, 29(3) :493–502, 1975.

UMK03. R. Ul-Mustafa and A. E. Kama. WDM network design with non-uniform traffic grooming. In *IEEE Optical Fiber Communication*, volume 2, pages 965–984, 2003.

Wal02. M. Walter. Évaluation et validation d'un logiciel de dimensionnement de réseaux. ESSI, DESS, 2002.

WCVM01. J. Wang, W. Cho, V. Rao Veemuri, and B. Mukherjee. Improved approaches for cost-effective traffic grooming in WDM ring networks : ILP formulations and single-hop and multihop connections. *IEEE/OSA Journal of Lightwave Technology*, 19(11) :1645–1653, November 2001.

WSM02. J. Wang, L. Sahasrabuddhe, and B. Mukherjee. Fault monitoring and restoration in optical WDM networks. In *National Fiber Optic Engineers Conference*, Dallas, USA, 2002. Telcordia Technologies.

Wu95. T. H. Wu. Emerging technologies for fiber network survivability. *IEEE Communications Magazine*, February 1995.

YJ02. S. Yuan and J. P. Jue. A shared protection routing algorithm for optical networks. *Optical Networks Magazine*, 3(3) :32–39, May/June 2002.

YLR99. J. Yates, J. Lacey, and M. Rumsewicz. Wavelength converters in dynamically reconfigurable WDM networks. *IEEE Communications Surveys & Tutorials*, 2(2) :2–25, 1999.

YOM03. S. Yao, C. Ou, and B. Mukherjee. Design of hybrid optical networks with waveband and electrical TDM switching. In *IEEE Globecom*, pages 2803–2808, San Francisco, CA, December 2003.

YRL98. J. Yates, M. Rumsewicz, and J. Lacey. Wavelength conversion in networks with differing link capacities. In *IEEE Globecom*, pages 2315–2320, Sydney, USA, December 1998.

ZGM03. A. Zsigri, A. Guitton, and M. Molnár. Two multicast algorithms for sparse splitting capable networks. In *IEEE Optical Fiber Communication*, volume 1, pages 3–21, 2003.

ZKW03. A. Zymolka, A. Koster, and R. Wessäly. Transparent optical network design with sparse wavelength conversion. In *IEEE Optical Fiber Communication*, volume 2, pages 61–80, 2003.

ZM02. K. Zhu and B. Mukherjee. Traffic grooming in an optical WDM mesh network. *IEEE Journal on Selected Areas in Communications*, 20(1) :122–133, January 2002.

ZQ00. X. Zhang and C. Qiao. An effective and comprehensive approach for traffic grooming and wavelength assignment in SONET/WDM rings. *IEEE/ACM Transactions on Networking*, 8(5) :608–617, October 2000.

ANNEXE A

Les réseaux optiques WDM

A.1. Exemples de pannes optiques

Nous montrons dans cette annexe quelques exemples d'incidents sur le réseau optique de FREE TELECOM[1], sur une période limitée à quelques mois. Ces incidents illustrent les différentes natures des pannes qui peuvent survenir sur un réseau optique. Les raisons des incidents et explications émanent directement de FREE TELECOM.

- 13 juillet 2004 de 00h00 à 06h00 : Swap de fibre dans le 94. Des travaux de génie civil nous obligent à devié nos fibres optiques dans le 94. Cela provoquera une coupure de service pendant 2h30 pour les clients ADSL degroupé dans le 94.
- 13 juillet 2004 de 09h30 à 10h30 : Travaux du tramway à Montpellier. En raison des travaux du tramway ces derniers jours, le réseau optique de la ville de Montpellier est amené à avoir des perturbations. Ceci a notamment engendré une coupure sur les sites de Rabelais, Rondelet, Lapeyronie, Facultés et Juvenal.
- 26 août 2004 depuis 13h10 : Incident sur le réseau optique de Montpellier. Pour une raison encore indéterminée, les sites de Juvenal, Rondelet, Lapeyronie Rabelais et Faculté sont actuellement isolés. La société de maintenance du réseau optique ainsi qu'une équipe de Free est sur place afin de solutionner le problème au plus vite.
- 28 septembre 2004 de 00h00 à 06h00 : Maintenance réseau optique Paris-Nord. Dans le cadre d'une réparation de fourreaux de fibres optiques endommagés dans les égouts de Paris, une opération de maintenance aura lieu cette nuit. Ceci occasionnera des perturbations (reroutage pour sécurisation) dans le réseau IP et voix ainsi que des coupures pour certains sites en ADSL dégroupés du nord de Paris (8, 9, 10, 16, 17, 18 et 19e arrondissement).
- 06 octobre 2004 de 04h15 à 08h15 : Incident sur le site de Bordeaux. En raison d'une défaillance matérielle sur un équipement de routage, le trafic sur les sites de Bordeaux et Toulouse a été fortement perturbé.
- 06 octobre 2004 de 11h10 à 14h30 : Incident sur la liaison Marseille-Nice. En raison d'un problème de manipulation de fibres optiques par notre fournisseur sur le site de St Maximin, le site de Nice a été isolé, impactant le trafic voix, RTC et ADSL.

[1] c.f. `http://support.free.fr/reseau/`

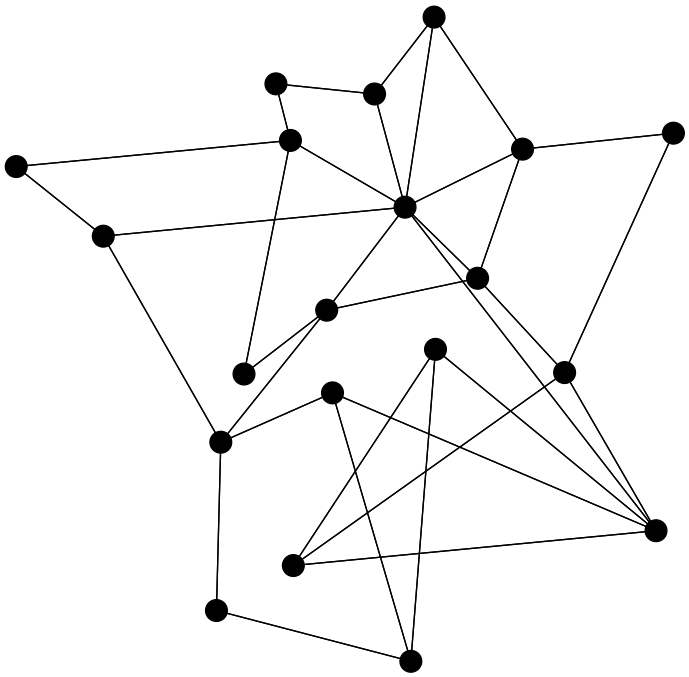
A.2. Réseaux d'étude

Cette annexe décrit succinctement les réseaux utilisés dans les différentes expérimentations de cette thèse. À l'inverse d'autres travaux, nous avons souhaité travailler le plus possible sur des instances de réseaux réels, et non pas sur des graphes aléatoires ou des graphes ayant des topologies régulières. Cette dernière remarque s'applique aussi à la distribution des requêtes. Cependant, nos tests incluent l'utilisation de deux grilles ayant des matrices de trafics régulières du type *All-to-All* et *Half-to-All*, provenant d'un travail plus global sur la génération d'instances de réseaux et de trafics [**Wal02**].

Pour chaque réseau, nous fournissons l'origine des données ainsi que le type de topologie, le nombre de nœuds et de liens de ce réseau. Nous précisons le nombre de requêtes (qui ne sont pas unitaires) et nous précisons la multiplicité des fibres du réseau (le nombre de longueurs d'onde par fibre), cette valeur pouvant être modifiée dans certaines expérimentations. Nous représentons ensuite la topologie de ces réseaux. Il faut noter que pour simplifier les graphiques, les liens sont représentés par des traits : ils sont en fait bidirectionnels. Dans la plupart des modèles évoqués dans cette thèse, les liens sont des arcs et les graphes des graphes orientés. Or, dans les instances issues de nos collaborations avec FRANCE TÉLÉCOM, tous les liens sont bidirectionnels : le même nombre de fibres sont disponibles dans chaque sens. Cette hypothèse quant à la symétrie des liens n'est pas forcément nécessaire et cela est d'autant plus vrai qu'une fibre optique peut être utilisée indépendamment dans les deux sens.

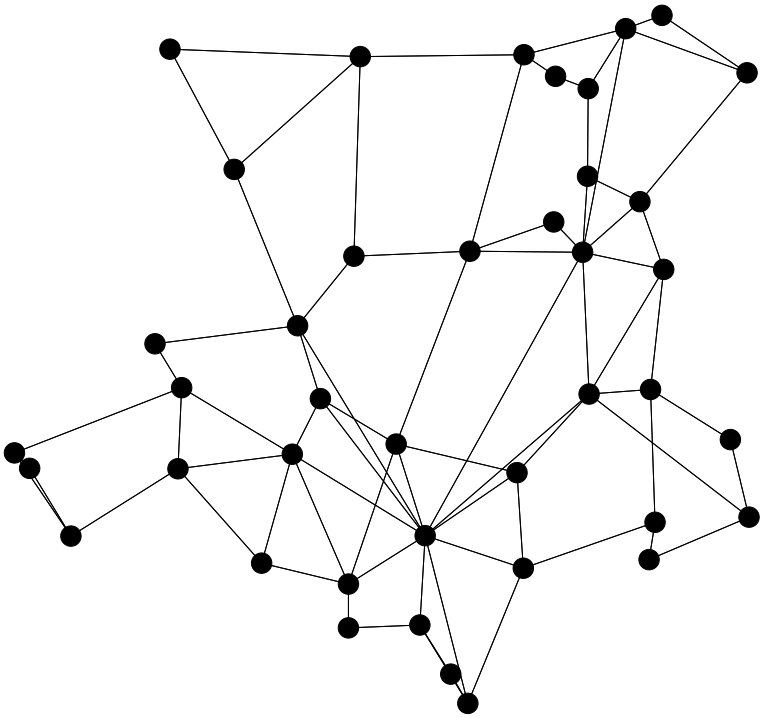
A.2.1. Réseau R_1

Nom : R_1
Topologie : Réelle
Source : France, fournie par FRANCE TÉLÉCOM [**IMR01**]
Nombre de nœuds : 20
Nombre de liens : 34
Nombre de requêtes : 118
Multiplicité en longueurs d'onde W **:** 50



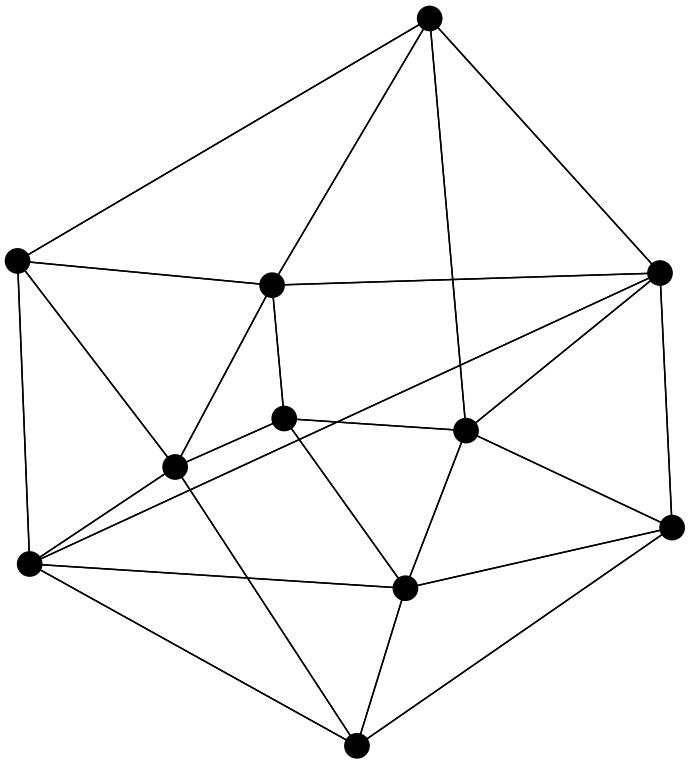
A.2.2. Réseau R_2

Nom : R_2
Topologie : Réelle
Source : France, fournie par FRANCE TÉLÉCOM [**Ba99**]
Nombre de nœuds : 41
Nombre de liens : 77
Nombre de requêtes : 134
Multiplicité en longueurs d'onde W **:** 32



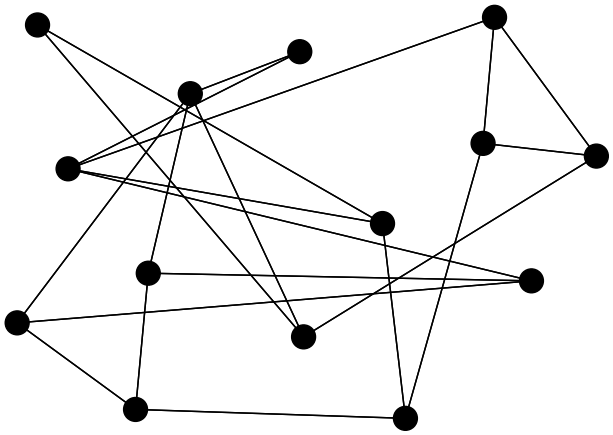
A.2.3. Réseau R_3

Nom : R_3
Topologie : Réelle
Source : Europe (Cost 239), fournie par FRANCE TÉLÉCOM [**Ba99**]
Nombre de nœuds : 11
Nombre de liens : 25
Nombre de requêtes : 110
Multiplicité en longueurs d'onde W **:** 20



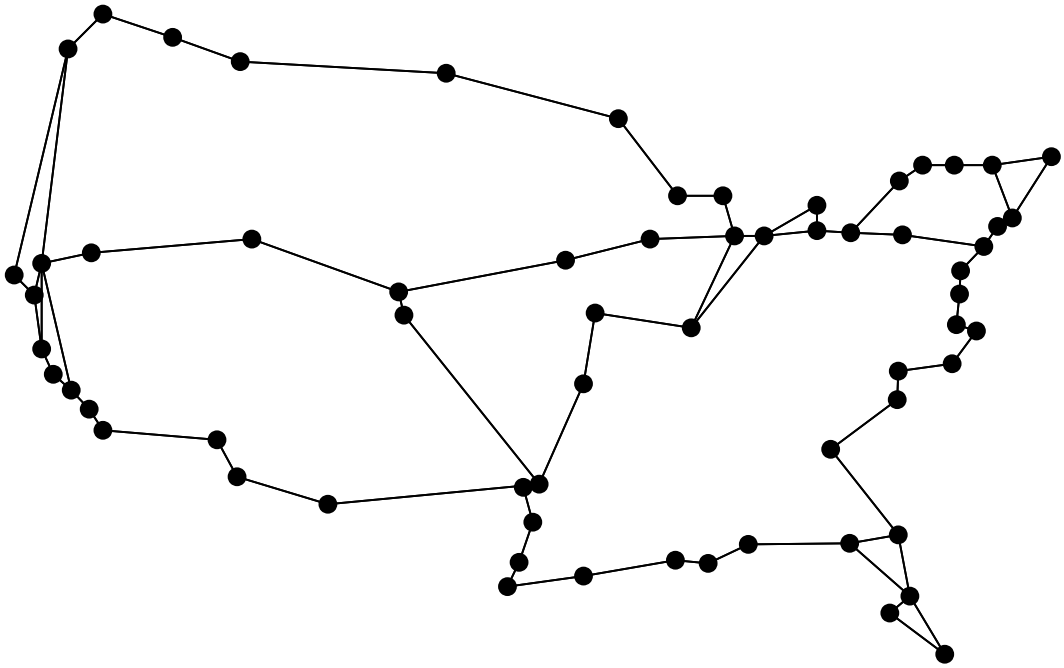
A.2.4. Réseau NSFNET

Nom : NSFNET
Topologie : Réelle
Source : NSFNET, décrite dans [**SS02, WSM02, YJ02**]
Nombre de nœuds : 14
Nombre de liens : 21
Nombre de requêtes : 140
Multiplicité en longueurs d'onde W **:** 36



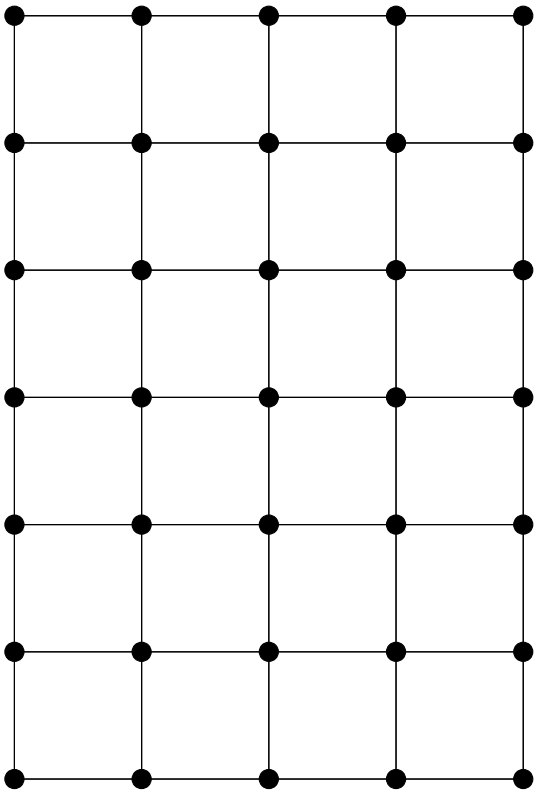
A.2.5. Réseau EU

Nom : EU
Topologie : Réelle
Source : Etats-Unis, créée à partir des informations de [**LE01**]
Nombre de nœuds : 64
Nombre de liens : 76
Nombre de requêtes : 190
Multiplicité en longueurs d'onde W **:** 36



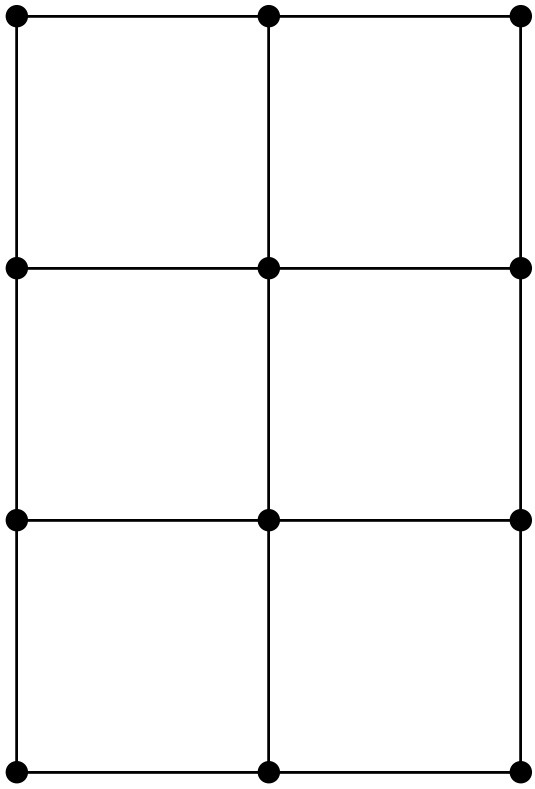
A.2.6. Réseau GRID_{35}

Nom : GRID_{35}
Topologie : Grille
Source : Générée dans [**Wal02**]
Nombre de nœuds : 35
Nombre de liens : 58
Nombre de requêtes : 1190
Multiplicité en longueurs d'onde W **:** 32



A.2.7. Réseau GRID_{12}

Nom : GRID_{12}
Topologie : Grille
Source : Générée dans [**Wal02**]
Nombre de nœuds : 12
Nombre de liens : 9
Nombre de requêtes : 132
Multiplicité en longueurs d'onde W **:** 32



ANNEXE B

Porto

B.1. Choix de la plate-forme et des langages

PORTO a été développé et testé sur des systèmes GNU/LINUX, et doit pouvoir être porté sans modification majeure sur tout système intégrant les outils de développement GNU (make, egcs), CPLEX SOLVER™ et la JDK de SUN. En pratique cela couvre également la plupart des systèmes UNIX et avec quelques efforts les systèmes MICROSOFT™ WINDOWS™. L'objectif initial n'était cependant pas de disperser les efforts sur ces problèmes de portage de logiciels mais bien de produire un outil d'expérimentation.

PORTO est divisé en deux sous-parties : l'interface graphique, écrite en JAVA™ et le moteur de calcul, écrit en C++. Ce choix, qui peut paraître étrange de prime abord, se justifie par les éléments suivants : les outils de développement des interfaces graphiques sont plus faciles à prototyper en JAVA™ (surtout pour un projet à durée limitée) et c'est donc naturellement JAVA™ qui nous a paru indispensable pour fournir une interface conviviale à PORTO. Ainsi, nous avons pu utiliser les outils de KOALA-GRAPHICS[1] du point de vue de l'interface graphique. Pour les calculs proprement dit, il nous a fallu interfacer notre code avec CPLEX SOLVER™, et les APIs fournies par ILOG SOLVER (API CONCERT) n'existait qu'en version C++. Cette version n'était compatible qu'avec le compilateur EGCS 2.1 ce qui a posé par la suite de nombreuses difficultés de compilation.

B.2. Interface principale

Nous allons décrire les fonctionnalités accessibles à travers l'interface en suivant un exemple. Une fois l'exécutable PORTO lancé, nous devons charger un réseau (au format décrit en 6.4), ce qui se fait naturellement depuis le menu file/open.

Nous ne présenterons pas en détail toutes les options de l'interface qui permettent de modifier la présentation du réseau, elles sont décrites dans les fichiers d'aide contenus dans la distribution du logiciel. On notera simplement que l'on peut sauvegarder les changements de présentation (couleurs, position des nœuds, ...) avec l'option *SaveSettings*. De plus, on peut sélectionner quels éléments doivent être affichés ou non dans la vue réseau par l'intermédiaire du menu *View*, ce qui permet parfois une meilleure visibilité. Par exemple, on affiche ou non les arcs du graphe (*Show Cable*), l'orientation des arcs (*Show Arrow*), les capacités disponibles sur les arcs (*Show free on cable*).

[1] cf. `http://koala.ilog.fr/graphics`

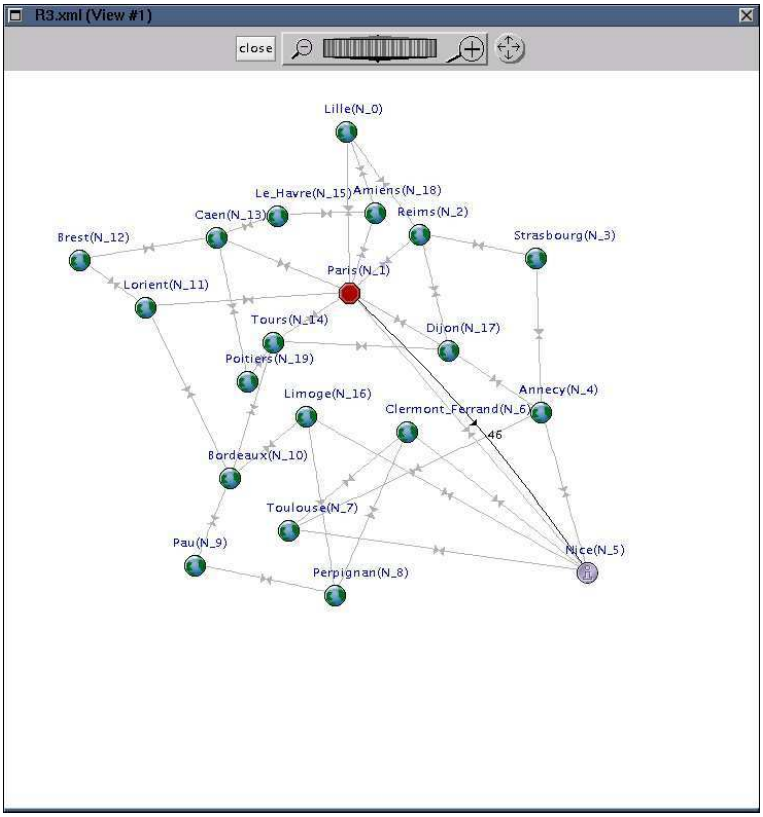


FIG. B.1 – Vue graphique du réseau R_1

Une fois le fichier XML chargé, on obtient la vue réseau de la figure B.1, sur laquelle nous avons mis en évidence une demande de 46 longueurs d'onde entre Paris et Nice. Les différents menus contextuels permettent de visualiser les demandes issues ou à destination d'un nœud. La fenêtre principale affiche toutes les demandes du réseau et permet de les examiner une à une, comme présenté dans la figure B.2. Les boutons *Main* et *Protection* permettent d'afficher les chemins principaux ou de protection quand ils existent (calcul effectué) et on peut éditer leurs couleurs. Le bouton *Set Path* permet de réserver manuellement des chemins pour une demande, ou seulement une partie de cette demande.

B.3. Exemple de déroulement d'un calcul de routage

B.3.1. Exécution d'un algorithme d'optimisation

Nous allons maintenant procéder au lancement des calculs d'optimisation pour le réseau R_1. La fenêtre qui s'affiche permet de sélectionner les différentes options de routage et de groupage qui ont été implémentées. Dans l'exemple de la figure B.3, on a choisi de calculer le routage en cochant la case *Routing*, et parmi les options disponibles on a choisi *multi* (pour multiroutage). Cela signifie qu'une demande peut être routée par un ensemble de chemins. Dans ce cas on ne propose pas de calculer des chemins de protection qui sont réservés au routage *single* (pour monoroutage). Notons que le résultat des calculs est par défaut enregistré dans un fichier différent de celui donné en entrée. Les algorithmes présents dans PORTO sont rappelés dans la

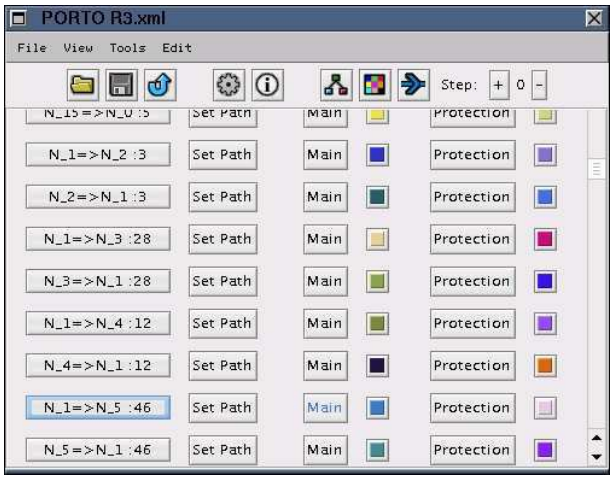


FIG. B.2 – Fenêtre principale de PORTO et vue de quelques requêtes

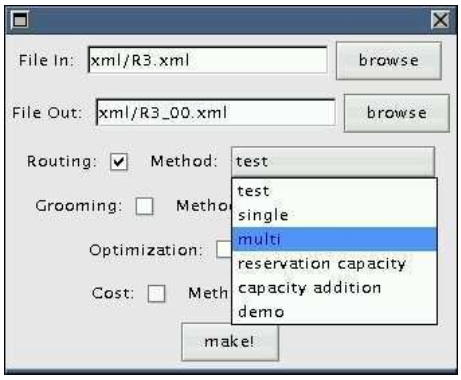


FIG. B.3 – Fenêtre de choix des options de routage et de groupage

partie consacrée aux résultats expérimentaux (section 6.5 et une explication sur les possibilités d'ajouter de nouveaux algorithmes de routage est donnée en section 6.2.

Ensuite on peut choisir une méthode de groupage parmi celles présentées en section 6.5 (figure B.4 et B.5). De la même façon que pour le routage il est facile d'ajouter de nouveaux algorithmes dans PORTO. Les champs *Max Cplex Solve Time* et *Optimality Tolerance* sont des paramètres pour le solveur CPLEX SOLVER™ utilisé dans le calcul du routage et qui indiquent de combien de temps en minutes le temps de calcul est borné ainsi que la tolérance de la distance de la solution par rapport à l'optimum. La case *Optimization value*, associée au groupage, indique le facteur de fermeture des conteneurs de niveau supérieur (fibres ou bandes). Un facteur égal à 0 indique que si une fibre contient ne serait-ce qu'une seule longueur d'onde, elle doit être fermée. Les autres longueurs d'onde non affectées sont "gaspillées".

Enfin, il faut cliquer sur le bouton *make* pour lancer l'optimisation. La fenêtre de la figure B.6 apparaît pour permettre à l'utilisateur de visualiser le bon déroulement ou non du programme. En particulier des messages d'erreur peuvent apparaître comme en cas de capacité

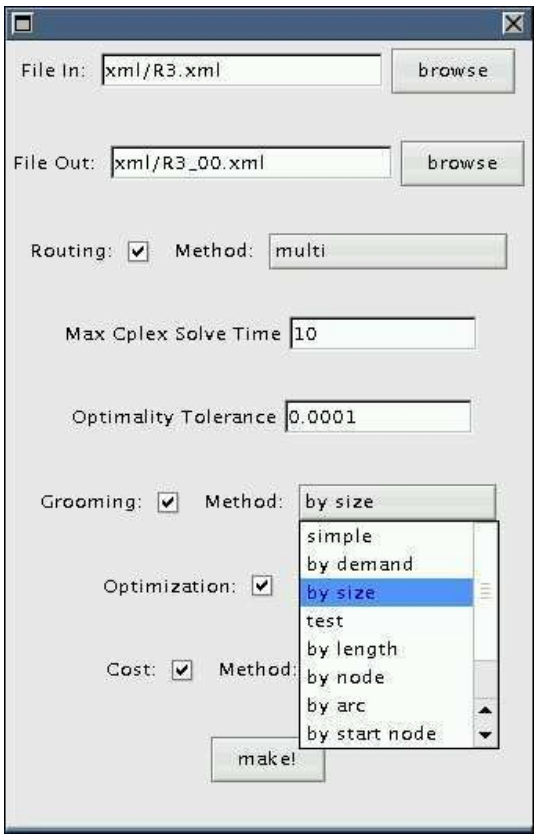


FIG. B.4 – Choix de l'option de groupage

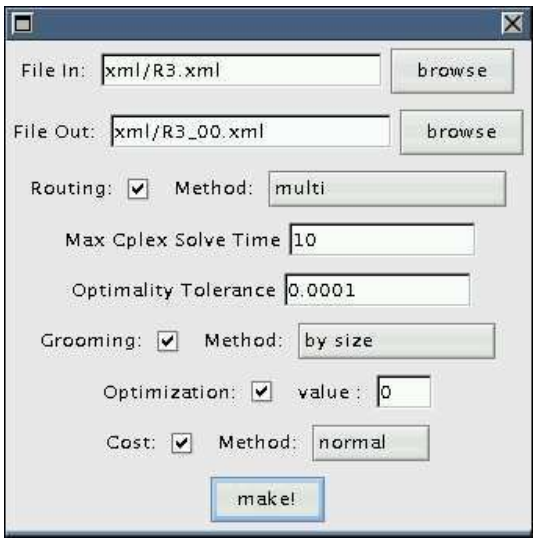


FIG. B.5 – Choix de la valeur du facteur de fermeture des conteneurs

insuffisante sur les nœuds ou bien en cas de dépassement de la taille mémoire pour des problèmes de grande taille. La fenêtre *Info* qui propose de charger le fichier résultat dans l'interface apparaît en cas de succès.

B.3.2. Visualisation du résultat

Une fois les calculs effectués avec succès, on charge le fichier obtenu et on peut visualiser plusieurs résultats. Nous montrons sur les figures suivantes les équipements et le brassage calculé en chaque nœud, l'affectation des demandes aux conteneurs optiques et le coût du réseau. Le contenu des câbles est présenté sur la figure B.7. Les fibres sont représentées en jaune, les bandes en bleu et les longueurs d'onde en rouge. Les bords latéraux indiquent que le conteneur est ouvert (ici pour la fibre à gauche), fermé (pour la fibre à droite) ou invisible car incluse dans un conteneur de plus haut niveau (comme pour les longueurs d'onde incluses dans les trois bandes du haut).

Sur la figure B.8 on voit la vue nœud obtenue en cliquant sur N_3. Les entrées sont à gauche et les sorties à droite. On peut suivre le routage des demandes qui sont issues, traversent ou arrivent en ce nœud. On peut remarquer sur cet exemple qu'une fibre fermée en provenance de N_2 arrive à destination (*Drop*). Pour la seule fibre en provenance de N_4, cinq longueurs d'onde arrivent à destination (*Drop*) et une autre contenue dans la même bande est multiplexée avec une longueur d'onde ajoutée en ce nœud (*Add*) pour sortir sur une des deux fibres à destination de N_2.

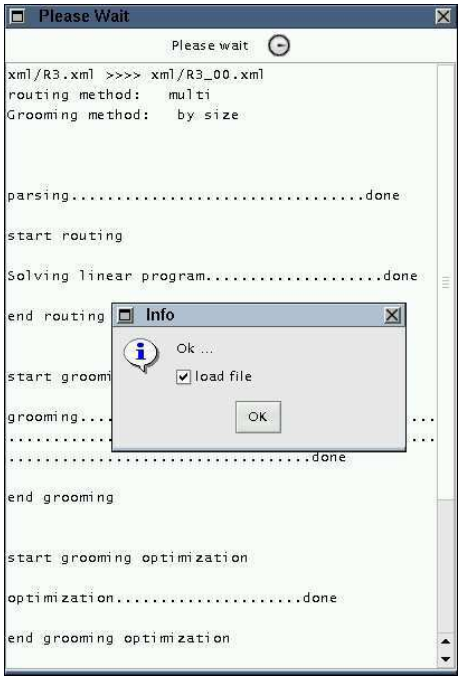


FIG. B.6 – Fenêtre d'information sur l'exécution du programme

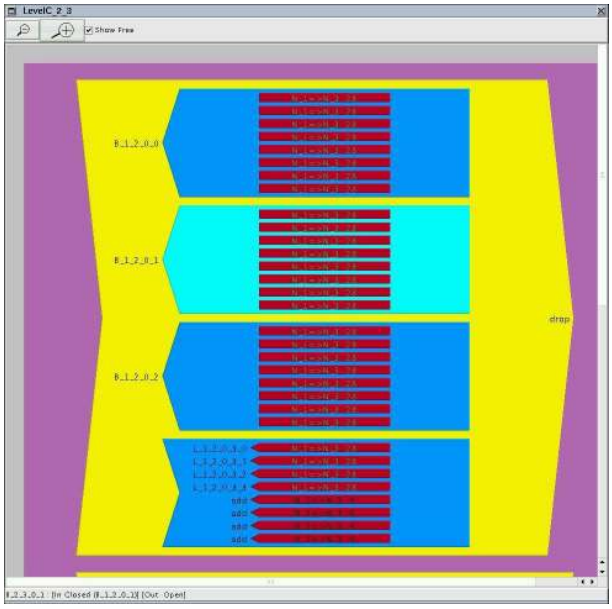


FIG. B.7 – Représentation d'un câble et de ses différents conteneurs

B.3.3. Format de données de sortie

Le format des entrées étant identique à celui des sorties, nous nous intéressons ici aux balises ou attributs qui apparaissent après les calculs de routage et groupage. La syntaxe précise

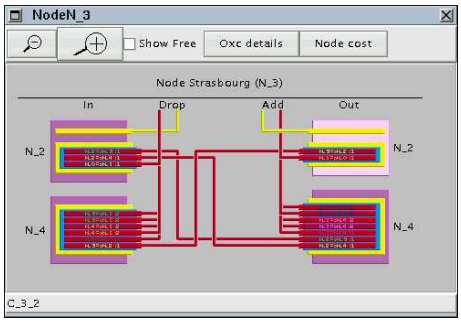


FIG. B.8 – Représentation des connexions dans un nœud du réseau

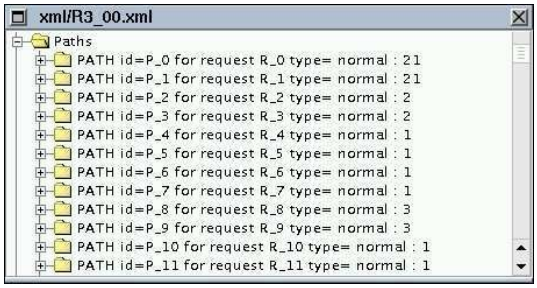


FIG. B.9 – Vue graphique des chemins

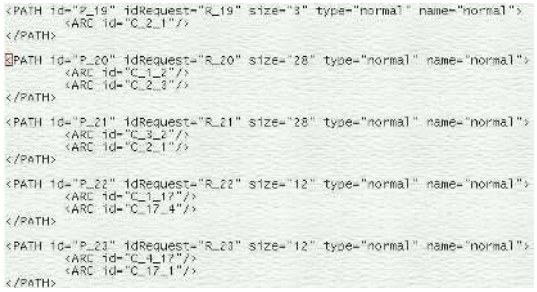


FIG. B.10 – Fichier XML des chemins

est détaillée dans le fichier *doc/exemplexml.xml* de la distribution. De plus, pour faciliter l'analyse des résultats et pour produire les courbes données en section 6.5, nous stockons un certain nombre de paramètres dans un fichier de type texte qui pourra facilement être parsé par des scripts ou de simples commandes shell.

Sur les figures B.9 et B.10 on voit la représentation dans l'interface et le code XML des chemins calculés lors de l'étape de routage. Les balises <PATH> sont des suites de balises <ARC>. Les balises <ARC> ont pour attribut l'identifiant id du câble correspondant.

Nous allons détailler maintenant le contenu des câbles après calcul. De nouveaux attributs sont requis. Pour les conteneurs élémentaires <LAMBDA> on doit indiquer si la longueur d'onde est affectée ou non à une demande. Sur l'exemple présenté sur les figures B.11 et B.12, il s'agit de la demande du nœud N_1 vers le nœud N_3 routée sur le chemin P_20. Les autres nouveaux attributs sont communs aux conteneurs supérieurs.

– InputState : open ou close ;
– Prev : l'id du conteneur de même niveau fermé en sortie qui précède celui-ci ou add ;
– OutputState : open ou close ;
– Next : l'id du conteneur de même niveau fermé en entrée qui succède à celui-ci ou drop.

Tous ces attributs sont optionnels.

À chaque calcul on produit aussi un fichier récapitulant les principaux résultats pour un usage statistique, comme montré dans le code source B.1. Cet exemple détaille les différents résultats concernant le nœud 0 qui permettent de calculer son coût. Ainsi, on connaît le degré par niveau de chaque élément de brassage pour ce nœud.

CODE SOURCE B.1 – Principaux résultats de l'optimisation

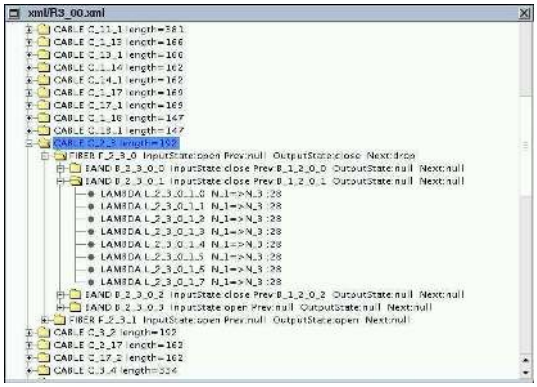


FIG. B.11 – Vue graphique des câbles

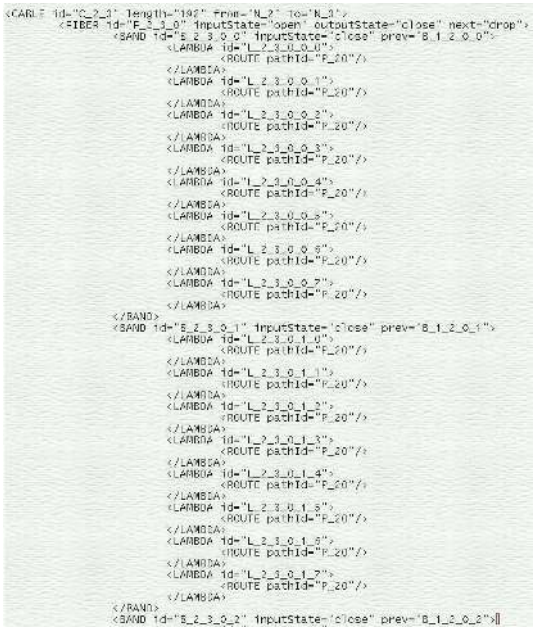


FIG. B.12 – Fichier XML des câbles

Numbers for result/caterpillar_multi_simple.xbar

```
edges: 7 cables x 2.
nodes: 8
total requests: 17
total FIBER: 28

Size of Xbar at node 0 :
 --FIBER: (in: 0) (out: 1) (add: 0) (drop: 0) (from BAND: 1) (to BAND: 0)
          (degree: 2) (length cost: 40) (xBar cost: 1914.07)
          (total cost: 1954.07)
 --BAND: (in: 0) (out: 2) (add: 1) (drop: 0) (from LAMBDA: 1) (to LAMBDA: 0)
         (degree: 4) (length cost: 0) (xBar cost: 2880.67)
         (total cost: 2880.67)
 --LAMBDA: (in: 0) (out: 7) (add: 7) (drop: 0) (degree: 14) (length cost: 0)
           (xBar cost: 3083.7) (total cost: 3083.7)
 --total for node: (degree: 20) (length cost: 40) (xbar cost: 7878.43)
                   (total cost: 7918.43)
```

ANNEXE C

Mascopt

C.1. Multiflot avec Cplex

Nous présentons succintement l'implémentation de la résolution avec CPLEX SOLVER™ du multiflot classique présenté au chapitre 2. Ce code fait aussi référence à la courte étude de cas de la section 7.5.

L'expression du multiflot se fait à travers l'interface CONCERT de CPLEX SOLVER™. Il s'agit d'un modèle objet représentant un programme linéaire. Ce modèle objet est constitué de variables, contraintes, fonctions objectifs possédant de nombreuses primitives d'accès. Il existe différentes façons d'accéder aux données du modèle permettant ainsi de concilier plusieurs vues du programme linéaire, comme par exemple sa forme matricielle. C'est cette forme qui a été privilégiée dans le chapitre 5, car elle est plus adéquate à la génération de colonnes. Dans le cas du multiflot, il est plus facile de manipuler les contraintes une à une.

En règle générale, on commence par générer l'ensemble des variables, raccrochées aux objets du modèle (par exemple, une variable CPLEX SOLVER™ pour un chemin du routage, s'il s'agit du modèle *arc-chemin*), puis on génère les contraintes et la fonction objectif. La résolution peut être entière ou flottante : le module d'optimisation utilisé par CPLEX SOLVER™ n'est alors pas le même : pour la résolution flottante, il utilise un algorithme du *simplexe* ou dérivé. Pour la résolution flottante il utilise un algorithme de *branchement* et de *coupes* (la "MIP" de CPLEX SOLVER™).

Les équations 2.1 et 2.2 du chapitre 2 sont implémentées par une bouche **while**, un premier bloc couvrant les requêtes, l'autre bloc couvrant les nœuds. Ensuite, pour chaque nœud, une autre boucle **while** couvre les arcs (ou arêtes, le code est générique) entrantes ou sortantes du nœud. Ce code nécessite la fonction **getVarEdgeReq(AbstractEdge e, Arc r)** qui retourne une variable CPLEX SOLVER™ pour une arête e ou une requête r, l'implémentation se faisant à l'aide d'un double niveau de **HashMap** imbriquées.

CODE SOURCE C.1 – Conservation du flot

```
/**
 * Adds a flow system on the network.
 */
public void createFlow(AbstractGraph g_, DiGraph requests_, IloCplex cplex)
{
  try
  {
    // For all requests
    Iterator itArcRequest = requests_.iterator();
    while (itArcRequest.hasNext())
```

```
    {
      // Getting request r
      Arc r = (Arc)itArcRequest.next();
      AbstractVertex n_start = r.getSource();
      AbstractVertex n_end = r.getTarget();
      double r_size = 0;

      // Getting the request size
      r_size = r.getDouValue(REQUEST_SIZE);

      // For all vertices
      Iterator itVertex = g_.getAbstractVertexSet().iterator();
      while (itVertex.hasNext())
      {
        // Getting vertex n
        AbstractVertex n = (AbstractVertex)itVertex.next();

        // Flow entering the vertex
        Iterator itEdge = n.getIn(g_).iterator();
        IloLinearNumExpr in = cplex_.linearNumExpr();
        while (itEdge.hasNext())
        {
          AbstractEdge edgeHere = (AbstractEdge)itEdge.next();
          in.addTerm(1.0, getVarEdgeReq(edgeHere,r));
        }

        // Flow exiting the vertex
        itEdge = n.getOut(g_).iterator();
        IloLinearNumExpr out = cplex_.linearNumExpr();
        while (itEdge.hasNext())
        {
          AbstractEdge edgeHere = (AbstractEdge)itEdge.next();
          out.addTerm(1.0, getVarEdgeReq(edgeHere,r));
        }

        // Case of the start
        if (n == n_start)
        {
          cplex_.addEq(out, r_size);
          cplex_.addEq(in, 0);
        }
        else if (n == n_end) // Case of the end
        {
          cplex_.addEq(in, r_size);
          cplex_.addEq(out, 0);
        }
        else // Normal case
        {
          cplex_.addEq(in, out);
        }
      }
    }

    // Solving
    cplex.solve();
  }
  catch (IloException exception)
  {
    System.err.println("Error in Cplex constraints " + exception);
  }
```

```
}
```

Après l'appel à CPLEX SOLVER™ pour la résolution, on doit lire la valeur des variables et l'on peut construire les chemins du flot. Cette partie est présentée dans le code source C.2.

CODE SOURCE C.2 – Build of a path for a request r

```
/**
 * Builds a paths covering a part of the flow.
 */
public AbstractPath buildPath(AbstractGraph g_, Arc r)
{
  AbstractVertex n_start = r.getSource();
  AbstractVertex n_end = r.getTarget();
  AbstractVertex n_current = n_start;
  double maxFlowAllocated = MAX_FLOW_PER_REQUEST;
  AbstractPath result = g_.getFactory().newAbstractPath(g_.getAbstractEdgeSet());

  // While the path has not reached the end vertex
  Iterator itArc = g_.getAbstractEdgeSet().iterator();
  while (n_current != n_end)
  {
    AbstractEdge candidate = null; AbstractEdge loopCandidate = null;
    double tryFlowAllocated = 0; double tryLoopFlowAllocated = 0;

    // We search for the best edge to continue the path
    Iterator itEdgeOut = n_current.getOut(g_).iterator();
    while (itEdgeOut.hasNext())
    {
      AbstractEdge eOut = (AbstractEdge)itEdgeOut.next();
      double cplexValue = getCplexValue(eOut,r); // Getting the result value of Cplex
      double reserved = eOut.getDouValue("pathConstructionReserved");
      AbstractVertex destOut = n_current.getConnected(eOut);

      // Searching the best candidate
      if (cplexValue - reserved > tryFlowAllocated && !result.getAbstractVertexSet().contains(destOut))
      {
        tryFlowAllocated = cplexValue - reserved;
        candidate = eOut;
      }

      // Searching the best loop candidate
      if (cplexValue - reserved > tryLoopFlowAllocated)
      {
        tryLoopFlowAllocated = cplexValue - reserved;
        loopCandidate = eOut;
      }
    }

    // If the path can be constructed
    if (candidate != null)
    {
      maxFlowAllocated = Math.min(maxFlowAllocated, tryFlowAllocated);
      result.concatAbstractEdge(candidate);
      n_current = n_current.getConnected(candidate);
    }
    else // Removing a found loop
    {
      n_current = this.removeLoop(result, loopCandidate, r);
    }
```

```
  }

  // Update of the reserved value on this path
  AbstractVertex vertex_covering = result.getAbstractStart();
  while (vertex_covering != result.getAbstractEnd())
  {
    AbstractEdge edge_parcours = result.nextAbstractEdge(vertex_covering);
    double reserved = edge_parcours.getDouValue("pathConstructionReserved");
    edge_parcours.setDouValue("pathConstructionReserved", reserved + maxFlowAllocated);
    vertex_covering = result.nextAbstractVertex(vertex_covering);
  }

  // Storing the flow value of this computed path
  result.setDouValue("flowAllocated", maxFlowAllocated);

  return result;
}
```

C.2. DTD

Cette section introduit la DTD qui valide les fichiers MGL de MASCOPT. Nous donnons la possibilité de désactiver la validation, mais nous pensons que l'utilisation de la DTD est une garantie forte pour la bonne formation des fichiers. Le code source C.3 décrit ce format natif. Si l'utilisateur souhaite étendre la grammaire, par exemple, pour ses objets spécialisés, il doit définir une DTD qui hérite de la DTD du format MGL. Nous montrons dans le code source C.4 un exemple d'héritage qui encode le comportement de valuation par contexte, présenté en section 7.6.3.

CODE SOURCE C.3 – DTD of MGL format : mgl_v1.2.dtd

```
<!ELEMENT OBJECTS (VERTICES, LINKS, SETS, PATHS?, GRAPHS)>

<!-- Groups -->

<!ELEMENT VERTICES (VERTEX*)>
<!ELEMENT LINKS (EDGE*, ARC*)>
<!ELEMENT SETS (VERTEX_SET*, EDGE_SET*, ARC_SET*)>
<!ELEMENT PATHS (CHAIN*, PATH*)>
<!ELEMENT GRAPHS (GRAPH*, DIGRAPH*)>

<!-- Infos -->

<!ELEMENT NAME (#PCDATA)>

<!ELEMENT VALUE (#PCDATA)>
<!ATTLIST VALUE type CDATA #REQUIRED>
<!ATTLIST VALUE dataType (String|Integer|Double) "String">

<!ELEMENT POSITION (X, Y)>

<!ELEMENT X (#PCDATA)>
<!ELEMENT Y (#PCDATA)>

<!-- Objects -->

<!ELEMENT VERTEX (NAME?, POSITION?, VALUE*)>
<!ATTLIST VERTEX id CDATA #REQUIRED>
```

```
<!ELEMENT EDGE (NAME?, VERTEX_REF, VERTEX_REF, VALUE*)>
<!ATTLIST EDGE id CDATA #REQUIRED>

<!ELEMENT ARC (NAME?, VERTEX_REF, VERTEX_REF, VALUE*)>
<!ATTLIST ARC id CDATA #REQUIRED>

<!-- Sets -->

<!ELEMENT VERTEX_SET (NAME?, VERTEX_REF*, VALUE*)>
<!ATTLIST VERTEX_SET id CDATA #REQUIRED>

<!ELEMENT EDGE_SET (NAME?, VERTEX_SET_REF, EDGE_REF*, VALUE*)>
<!ATTLIST EDGE_SET id CDATA #REQUIRED>

<!ELEMENT ARC_SET (NAME?, VERTEX_SET_REF, ARC_REF*, VALUE*)>
<!ATTLIST ARC_SET id CDATA #REQUIRED>

<!-- Paths -->

<!ELEMENT CHAIN (NAME?, EDGE_REF*, VALUE*)>
<!ATTLIST CHAIN id CDATA #REQUIRED>

<!ELEMENT PATH (NAME?, ARC_REF*, VALUE*)>
<!ATTLIST PATH id CDATA #REQUIRED>

<!-- Graphs -->

<!ELEMENT GRAPH (NAME?, VERTEX_SET_REF, EDGE_SET_REF, VALUE*)>
<!ATTLIST GRAPH id CDATA #REQUIRED>

<!ELEMENT DIGRAPH (NAME?, VERTEX_SET_REF, ARC_SET_REF, VALUE*)>
<!ATTLIST DIGRAPH id CDATA #REQUIRED>

<!-- Pointers -->

<!ELEMENT VERTEX_REF EMPTY>
<!ATTLIST VERTEX_REF idref CDATA #REQUIRED>

<!ELEMENT EDGE_REF EMPTY>
<!ATTLIST EDGE_REF idref CDATA #REQUIRED>

<!ELEMENT ARC_REF EMPTY>
<!ATTLIST ARC_REF idref CDATA #REQUIRED>

<!ELEMENT VERTEX_SET_REF EMPTY>
<!ATTLIST VERTEX_SET_REF idref CDATA #REQUIRED>

<!ELEMENT EDGE_SET_REF EMPTY>
<!ATTLIST EDGE_SET_REF idref CDATA #REQUIRED>

<!ELEMENT ARC_SET_REF EMPTY>
<!ATTLIST ARC_SET_REF idref CDATA #REQUIRED>

<!ELEMENT SUPER_SET_REF EMPTY>
<!ATTLIST SUPER_SET_REF idref CDATA #REQUIRED>

<!ELEMENT SUPER_GRAPH_REF EMPTY>
<!ATTLIST SUPER_GRAPH_REF idref CDATA #REQUIRED>
```

CODE SOURCE C.4 – DTD of MGX format : mgx_v0.3.dtd

```
<!ENTITY % mgldtd SYSTEM "mgl_v1.2.dtd">

mgldtd;

<!-- Extended -->

<!ATTLIST VALUE context CDATA #IMPLIED >
```

C.3. Diagramme UML de certains packages de Mascopt

C.3.1. Diagramme des classes abstraites de graphes

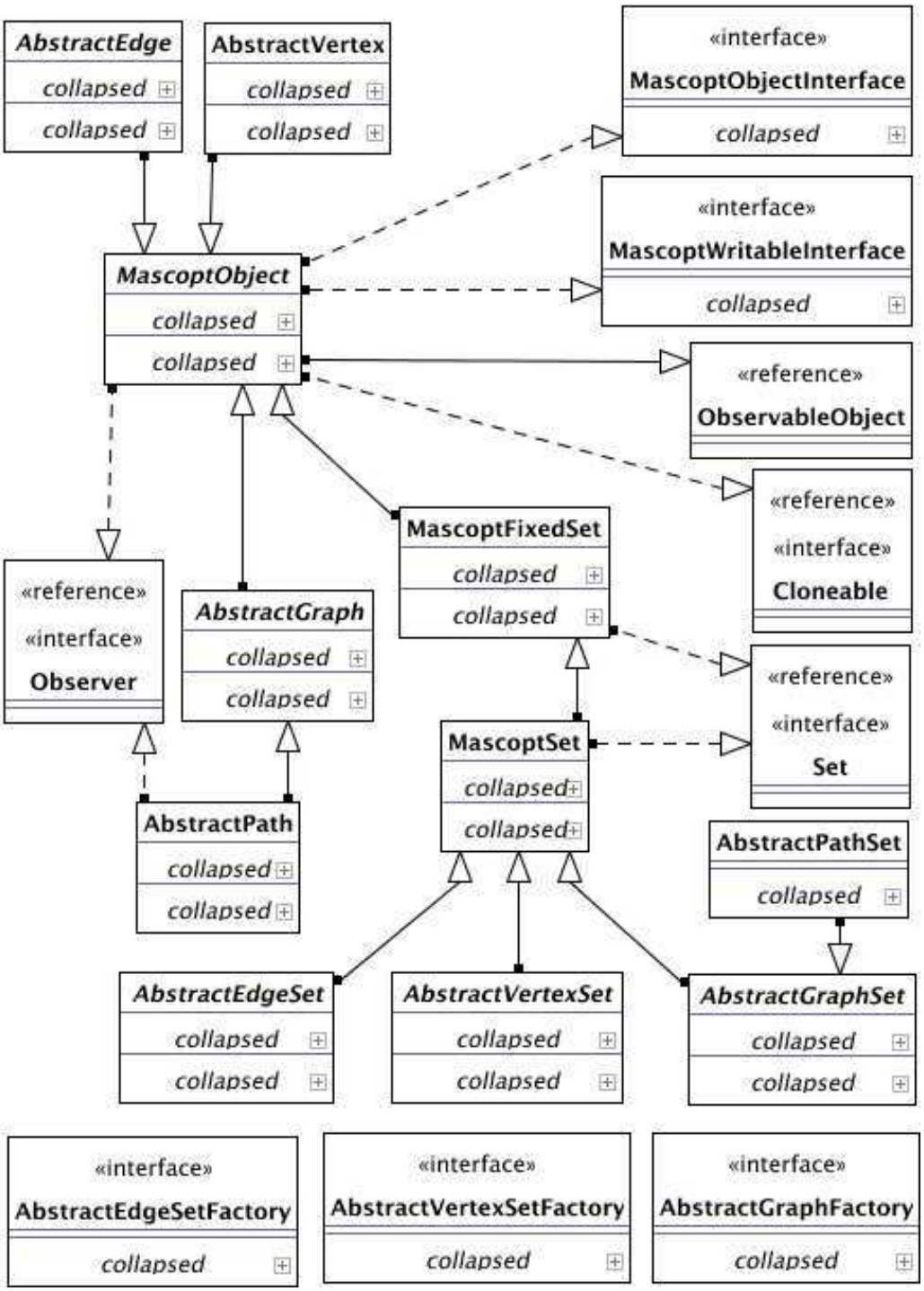


FIG. C.1 – Diagramme UML des classes abstraites de la partie graphe de MASCOPT

C.3.2. Diagramme des classes non abstraites de graphes

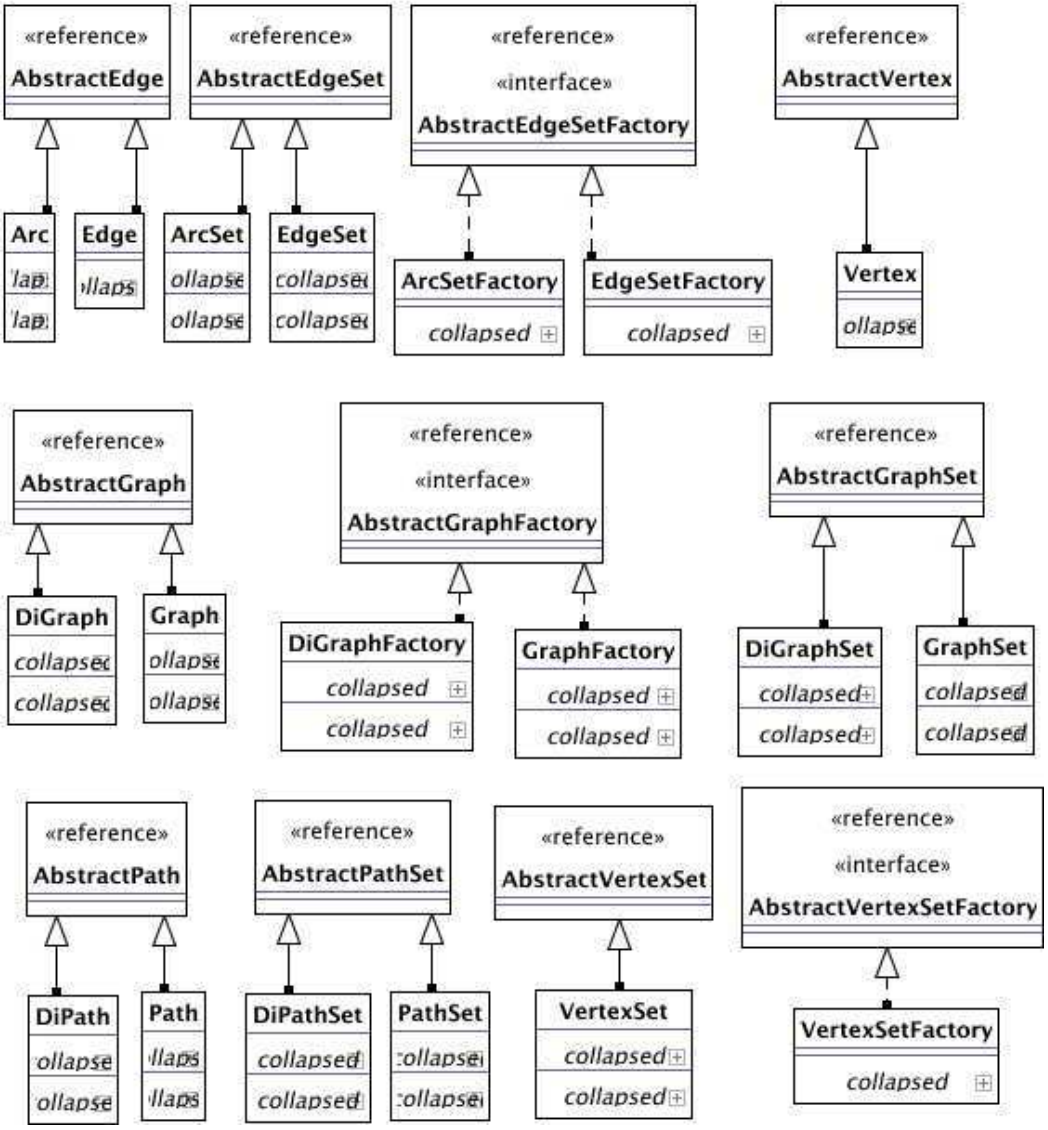


FIG. C.2 – Diagramme UML des classes non abstraites de la partie graphe de MASCOPT

Résumé

Dans cette thèse, nous nous intéressons aux problèmes d'optimisation dans les réseaux de télécommunication. Un premier objectif consiste à identifier les problèmes spécifiques aux réseaux optiques et satellitaires, et à présenter des contributions pour l'optimisation des ressources de ces réseaux. Le second objectif est de présenter une contribution logicielle pour la conception et l'optimisation de réseaux.

La première partie débute par la présentation des réseaux optiques WDM. Nous abordons ensuite les modèles pour les réseaux optiques et satellitaires et proposons des méthodes algorithmiques nouvelles pour optimiser l'allocation des ressources de ces réseaux. Nous traitons ainsi le problème du routage, du groupage et de la protection des réseaux WDM successivement dans trois chapitres puis nous nous intéressons à un algorithme dédié à l'allocation de fréquences dans les réseaux satellitaires. Enfin, pour chaque problème, nous présentons des résultats expérimentaux sur des instances de réseaux réels.

La deuxième partie de cette thèse présente les développements logiciels qui ont été entrepris. Le premier chapitre présente le logiciel PORTO dédié à la résolution de problèmes de routage, groupage et protection dans des réseaux optiques utilisant trois niveaux de brassage. Dans un second chapitre nous présentons le logiciel MASCOPT, une bibliothèque d'optimisation pour le domaine des graphes et des réseaux qui a servi notamment à réaliser les expérimentations présentées dans la première partie.

Mots clefs : optimisation combinatoire, programmation linéaire entière, expérimentations, réseaux optiques.

Abstract

This thesis deals with optimization problems in telecommunication networks. Our first goal consists in identifying the specific problems in optical and satellite networks and in presenting our contributions for optimizing the network resources. The second goal consist in presenting the developed softwares for the design and optimizations of networks.

The first part begins with the presentation of WDM networks. We introduce the models for optical and satellites networks and we propose new algorithmic methods for optimizing resources allocation in these networks. We deal with different problems, routing, grooming and survivability of WDM networks in the three following chapters. Then, we focus on a specific algorithm for allocating frequencies in a satellite network. For each issue, we present our experiments using real network instances.

The second part of this thesis introduces the software developments that have been achieved. The first chapter presents the PORTO software which deals with solving routing, grooming and survivability problems in optical networks that uses three hierarchical levels of switching. In the second chapter, we detail the MASCOPT software, a library dedicated to graphs and networks optimization which was used for experiments of the first part.

Keywords : combinatorial optimization, linear programming, experiments, optical networks.