MINISTERE DE L'ENSEIGNEMENT SUPERIEUR DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE TUNIS EL MANAR
المدرسة الوطنية للمهندسين بتونس
ÉCOLE NATIONALE D'INGENIEURS DE TUNIS

DEPARTEMENT TECHNOLOGIES DE L'INFORMATION ET DES COMMUNICATIONS

PROJET DE FIN D'ETUDES

PRESENTE A

L'ECOLE NATIONALE D'INGENIEURS DE TUNIS

POUR OBTENIR LE

DIPLOME NATIONAL D'INGENIEUR EN TELECOMMUNICATIONS

PAR

BELHARETH SONIA

ROUTING RECONFIGURATION IN WDM NETWORKS

EN COLLABORATION AVEC :

EQUIPE MASCOTTE DE L'INRIA-SOPHIA-ANTIPOLIS

ENCADRE(E) PAR :

M. DAVID COUDERT
M. MOHAMED KOUBAA

ANNEE UNIVERSITAIRE : 2009-2010

B.P. 37 Le Belvédère 1002 Tunis - Tunisie الجمهورية التونسية ص ب 37 البلفدار 1002 تونس –
Courriel : enit@enit.rnu.tn البريد الالكتروني: Fax : +216872729 الفاكس: Téléphone : +216718747 00 الهاتف:

Résumé

Les réseaux WDM (Wavelength-Division-Multiplexing WDM) sont des réseaux composés d'un ensemble de nœuds interconnectés entre eux par des fibres [7]. Dans un réseau WDM, la reconfiguration du reroutage est un problème fondamental, c'est le fait de changer le reconfiguration du réseau en modifiant les routes de certaines requêtes, soit suite à des évolutions de trafic ou pour interdire l'utilisation de certaines ressources [1]. Le but de notre stage était d'implémenter des heuristiques pour résoudre le problème de reconfiguration qui minimisent le coût de reroutage et le nombre d'interruptions des requêtes. Le coût proposé est un modèle issu de contraintes physiques effectivement observées dans les réseaux. Nous avons étudié particulièrement deux cas, le premier cas est lorsque les ressources sur chaque lien ne sont pas limitées, elles sont toujours disponibles lors du reroutage. Lors de ce premier cas nous avons commencé par étudier théoriquement un cas particulier des graphes [8], les anneaux orientés symétriques puis proposé et comparé plusieurs heuristiques. Le deuxième cas intègre la notion de rareté de ressources, dans ce cas des requêtes doivent être interrompues temporairement. Nous avons adopté les heuristiques obtenues dans le premier cas à ce contexte.

Mots clés :
Algorithmique, théorie de graphe, langages de programmation, optimisation combinatoire

Dédicace

A mon père, ma mère
Pour tout leur soutient, leurs sacrifices et leur amour
A mes frères
Avec tous mes souhaits d'un grand succès dans leur vie
A mes sœurs
Que Dieu les préserve de toute peine et de tout malheur
A mes chères neveux
Que Dieu les protèges
A tous mes amis
A tous ceux que j'aime et tous ceux qui me sont chers
Je dédie ce travail...
Sonia

"Si j'ai pu voir plus loin que les autres, c'est parce que je m'appuyais sur les épaules de géants"

Isaac Newton

Avant propos

Ce travail a été effectué dans le cadre du projet de fin d'études du cycle d'Ingénieur en Télécommunications à l'École National ingénieurs de Tunis (ENIT). Ce projet a été réalisé au sein de l'INRIA, Sophia Antipolis, France (Institut national de recherche en informatique et automatique), à l'équipe Mascotte (Méthodes algorithmiques, simulation, combinatoire et optimisation des télécommunications).

L'objectif de l'équipe Mascotte est de développer des méthodes et outils algorithmiques qui s'appliquent en particulier à la conception de réseaux de télécommunications. La réalisation de cet objectif implique la poursuite de recherches dans les domaines de l'algorithmique, de la simulation, des mathématiques discrètes et de l'optimisation combinatoire. Elle s'intéresse plus particulièrement à l'Algorithmique des communications pour des problèmes tels que le dimensionnement de réseaux (optiques WDM, MPLS, embarqués, radio WiFi WiMax et satellites).

Au terme de ce travail, je souhaite remercier en premier lieu Jean-Claude Bermond pour m'avoir donné la chance de travailler au sein de l'équipe Mascotte. J'adresse mes vifs remerciements et mes respects les plus profonds à mon encadrant M. David Coudert, Chargé de Recherche à INRIA, pour m'avoir accueillie. Je tiens aussi à le remercier pour ses précieux et judicieux conseils scientifiques et moraux, sa générosité et ses encouragements incessants qu'il m'a procuré tout au long de mon stage. Je tiens fort à exprimer ma vive reconnaissance et mes remerciements à Nicolas Nisse, Chargé de Recherche à Mascotte, Philippe Giabbanelli, Issam Tahiri, Dorian Mazauric et Nathann Cohen, doctorants à Mascotte INRIA, pour l'aide inestimable, la disponibilité, les discussions constructives et surtout leur encouragement répété.

Je tiens à exprimer ma gratitude envers M. Mohamed Koubàa, Maître assistant et directeur du département Technologies de l'information et des communications à ENIT, pour le suivi de mes travaux, pour son soutien, ses

encouragements, sa disponibilité et ses précieux conseils.

Je remercie également les membres de jury pour avoir accepté d'évaluer ce travail.

Mes vifs remerciements s'adressent à tous les membres de l'équipe Mascotte pour leurs contributions à me procurer les conditions favorables pour le déroulement de mon stage et pour l'ambiance de travail agréable que j'ai trouvée au sein de cette équipe.

Mes remerciements s'adressent également à tous ceux qui ont contribué à ma formation, particulièrement les enseignants de l'École National d'ingénieurs de Tunis.

Table des matières

Table des figures

Table des algorithmes

Chapitre 1

Introduction Générale

L'apparition des systèmes de transmission à base de fibres optiques a beaucoup influencé le monde des télécommunications et la majorité des opérateurs ont migré vers les réseaux de communication optiques. Le facteur principal d'un tel choix réside dans le fait que les réseaux WDM optiques constituent la meilleure solution économique pour mettre en place des réseaux hauts débits pouvant supporter la croissance du trafic [7] [9] .

La reconfiguration de routage est un problème fondamental dans les réseaux de télécommunication. Son introduction dans les réseaux WDM a procuré une adaptabilité accrue des réseaux à un type de trafic de plus en plus variable. En pratique, chaque opérateur cherche un réseau optique moins couteux, plus flexible et reconfigurable, dont la capacité peut être aisément augmentée avec l'accroissement du trafic.

Pour ce type de réseau, une contrainte supplémentaire s'ajoute lors de la reconfiguration : la rareté des ressources (i.e., canaux WDM nécessaires pour le reroutage du trafic). Ceci permet de distinguer deux cas, le premier cas est lorsqu'il s'agit d'un réseau doté de ressources illimitées. C'est-à-dire qu'il n'y a pas de contrainte sur le nombre de longueurs d'onde et lors du reroutage on a toujours des ressources disponibles. Tandis que le deuxième cas est le cas des ressources limitées ce qui génère une dépendance entre les requêtes lors du passage du premier routage au routage final.

Dans la littérature, plusieurs travaux ont étudié le problème de la reconfiguration du routage. Chacun d'entre eux possède un objectif spécifique à ses besoins. C'est-à-dire une fonction de coût propre au problème. Dans notre cas et dans un souci de présenter un module de reconfiguration performant et conforme à la réalité, nous avons fixé une fonction de coût simplifiée mais tirée de contraintes physiques qui n'avaient pas encore été étudiées. Notre objectif est de rerouter les demandes avec le moindre coût et le nombre minimum d'interruptions lors de la reconfiguration.

Ce travail est composé de trois chapitres. Le premier chapitre est réservé à la présentation du contexte et de la problématique. Il offre ainsi les notions nécessaires en théorie du graphes, réseaux WDM et reconfiguration, pour appréhender ce travail. C'est à ce niveau que nous effectuons une description des travaux de reconfiguration du routage existant. Le deuxième chapitre est divisé en deux sections. La première est réservée à l'étude théorique du cas particulier des anneaux orientés symétriques. Dans la deuxième section nous étudions le problème de la reconfiguration dans le cas des ressources illimitées. Nous détaillons le principe des approches de reconfiguration du routage qui ont été développées dans ce contexte et nous les comparons par le biais des simulations. Le dernier chapitre présente les différentes approches de reconfiguration développées suite à l'intégration de la notion de la rareté des ressources (lorsque des interruptions sont nécessaires). Ceci est réalisé grâce à l'étude d'un graphe auxiliaire, le graphe de dépendance [2]. Enfin nous dressons une synthèse des résultats et des enjeux futurs.

Chapitre 2

Contexte et Problème

2.1 Problématique, Objectif et Contraintes

Nous travaillons sur un réseau optique utilisant la technologie du multiplexage en longueur d'onde (Wavelength-Division-Multiplexing WDM). Un réseau WDM est un réseau composé d'un ensemble de nœuds interconnectés entre eux par des fibres optiques [1] [7]. Soit un exemple de réseau WDM modélisé par un graphe anneau orienté symétrique présenté par la Figure 2.1. La capacité, c'est-à-dire le nombre de longueurs d'onde, sur chaque arc du réseau est égale à 1.

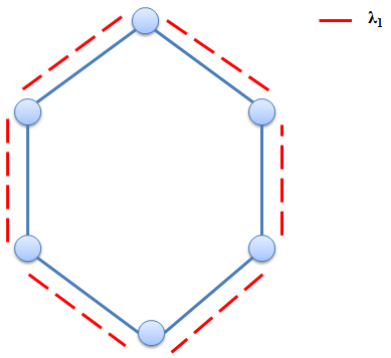

FIGURE 2.1 – Réseau WDM avec une capacité = 1 sur chaque arc

Une fois que le réseau WDM est mis en place, il sera utilisé pour router une quantité du trafic propre à un ensemble des demandes. Une demande est une paire de sommets, à laquelle nous associons un chemin optique dans le réseau, c'est-à-dire un chemin et une longueur d'onde de bout-en-bout, avec

la contrainte que deux chemins optiques partageant une même fibre optique doivent avoir des longueurs d'ondes différentes. L'ensemble de ces chemins optique est appelé routage.

Prenant l'exemple de 3 demandes a, b et c qui sont modélisées respectivement par les paires des sommets (2, 4), (3, 1) et (1, 4). Pour router ces demandes, on a besoin de déterminer pour chaque demande une route et une longueur d'onde pour le routage de son trafic. La Figure 2.2 présente un exemple de routage de ces demandes dans le réseau anneau. Ici, il n'y a pas de conflit entre les chemins. Nous pouvons attribuer la même longueur d'onde à toutes les requêtes.

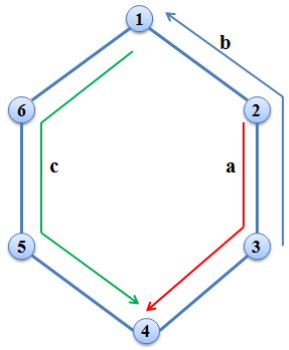

FIGURE 2.2 – Routage des demandes sur l'anneau

La reconfiguration du routage est le passage d'un routage courant R1 à un autre routage R2, en déplaçant successivement chaque requête. Soit l'exemple deux routages R1 et R2 présentées au niveau de la Figure 2.3. Pour passer de R1 à R2, on peut déplacer c, puis a et enfin b, mais cela demande à l'arête 6-5 "d'accueillir" 3 longueurs d'onde à un moment donné. Une autre façons de faire serait d'interrompre b, déplacer a puis c. La charge est alors diminuée au dépend de la qualité de service (QoS).

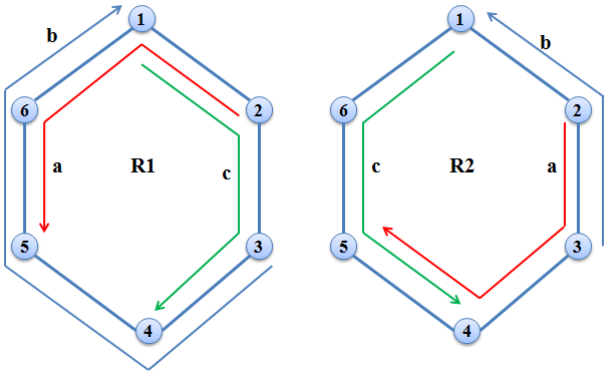

FIGURE 2.3 – Deux routages R1 et R2

Deux cas de traitement se présentent. Le premier cas est lorsqu'on n'a pas une contrainte de ressources, c'est-à-dire qu'il n'y a pas de contrainte sur le nombre de longueurs d'onde. Dans ce cas, lors du reroutage, on a toujours des ressources disponibles, donc on n'aura pas besoin d'interruptions.

Les contraintes physiques prises en compte, sont liées au temps nécessaire à la réservation d'un chemin et d'une longueur d'onde de bout-en-bout dans le réseau. Ce temps dépend du nombre de longueurs d'onde déjà utilisées sur les fibres optiques considérées. Donc, lors de passage d'un routage à un autre il y des adaptations qui doivent se faire sur les nouvelles routes dans le réseau. Pour simplifier l'étude de problème et se rapprocher de la réalité, nous avons fixé comme fonction de coût :

$$\sum_{e \epsilon R_f} (C(e))^{\alpha}$$

où C(e) représente la charge sur un lien e $\in R_f$, c'est-à-dire le nombre de longueurs d'onde qui sont déjà utilisées sur l'arête e, et α est une variable aléatoire comprise entre 0 et 10.

La fonction objectif du premier cas est la minimisation de la fonction de coût associée au routage d'une requête vers sa nouvelle route (i.e., route finale) R_f.

Le deuxième cas de traitement est lorsque les ressources sont limitées. Dans ce cas, des requêtes doivent être interrompues temporairement (make-before-break [1]) de façon à réaliser le reroutage. Pour déterminer quelles requêtes doivent être interrompues et pour minimiser les perturbations, nous

utilisons l'approche proposée en [6] et la notion de graphe de dépendance. Ce graphe présente la dépendance entre les demandes lors du passage de R1 à R2. Il est construit de la manière suivante, un arc de b vers a dans le graphe de dépendance signifie que la demande b passe dans le routage R2 par un ou plusieurs arc de la route de a dans R1 et sur la même longueur d'onde. En d'autre terme, b veut utiliser dans R2 des ressources utilisées par a dans R1. Le graphe de dépendance obtenu à partir de R1 et R2 (Figure 2.3) est illustré au niveau de la Figure 2.4.

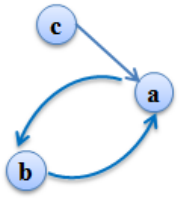

FIGURE 2.4 – Graphe de dépendance de R1 et R2

Dans le cas où le graphe de dépendance contient un ou plusieurs cycles, il y a interblocage et il est nécessaire d'interrompre une requête pour résoudre le blocage. Nous visons donc à effectuer le passage de R1 à R2 tout en minimisant d'une part les interruptions du trafic au sein du graphe du dépendance, et d'autre part la fonction de coût.

Le fait d'interrompre temporairement une ou plusieurs requêtes est ce qu'on appelle break-before-make, c'est-à-dire que la requête est interrompue jusqu'à la libération des ressources dont elle a besoin dans sa route finale. En effet, pour passer du routage initial au routage final, deux types de commandes seront utilisées : make-before-break et break-before-make qui ont été normalisées par GMPLS (Generalized Multi-Protocol Label Switching) [1] [2].

La notion make-before-break consiste à la réservation des ressources nécessaires à une requête dans sa nouvelle route, puis à y placer le trafic avant de libérer les ressources utilisées par la route initiale. En ce qui concerne la notion break-before-make, elle consiste à interrompre la connexion et à libérer les ressources qui étaient utilisées, ensuite réserver les ressources de la nouvelle route avant de replacer le trafic.

Notre contribution consiste en un ensemble de résultats théoriques dans le cas de topologies simples et d'heuristiques, pour déterminer l'ordre dans

lequel les requêtes doivent être reroutées afin minimiser le coût.

Pour l'étude du problème de reconfiguration du routage, nous modélisons le réseau WDM par un graphe G (V, E) orienté symétrique. Soit un nombre défini de longueurs d'onde λ sur chaque arête e $\in$ E de G. Le réseau WDM considéré est un réseau transparent (voir section 2.3.2).

Objectifs

A partir d'une instance I (i.e., ensemble des requêtes à rerouter) et des routages initial et final, l'objectif est de trouver l'ordre de reroutage des requêtes qui minimise la fonction de coût. Dans le cas des ressources limitées, nous cherchons en premier lieu à minimiser les interruptions du trafic (utilisant l'approche de [2]), nous cherchons ensuite un ordre de traitement qui minimise de même la fonction de coût.

2.2 Contributions

Plusieurs algorithmes de reconfiguration de reroutage sont développés, chacun possède une fonction objectif bien définit (voir Section 2.3.3).
Nous avons fixé une fonction objectif qui regroupe à la fois la minimisation du nombre de requêtes devant être temporairement interrompus et la minimisation du coût de reroutage.
Nous avons consacré la section 2.3 à la présentation de quelques notions utilisées dans ce travail : la théorie des graphes , les réseaux WDM ainsi qu'un aperçu sur les algorithmes de reconfiguration de routage qui existent dans la littérature. Nous avons montrés dans la section 3.2, que le problème étudié est NP-difficile [10]. Une partie théorique est ensuite mise en place au niveau de la section 3.3. Elle traite complètement un cas particulier des graphes, anneau orienté symétrique. Nous citons dans le reste du chapitre 3 les approches de reconfiguration du routage développées pour le cas des ressources illimitées. Nous avons développé aussi des approches de reconfiguration qui traitent le graphe de dépendance [1], dans le cas où les ressources sont limitées. Ces approches font l'objet du chapitre 4.

2.3 Notions utilisées

2.3.1 Théorie des graphes

La théorie des graphes constitue aujourd'hui un recueil de connaissances très important, elle est née en 1736 avec la communication d'Euler (1707-1783) dans laquelle il proposait une solution au célèbre problème des ponts de Konigsberg (Euler, 1736) [8].

Un graphe s'avère être une représentation abstraite de notions concrètes. Il peut être vu comme un ensemble de points et des lignes reliant certains points. A l'aide d'une telle notion on arrive à modéliser des situations concrètes très variées. Leurs applications sont particulièrement importantes en mathématique, en informatique, notamment en réseaux de communications et théorie de la complexité, etc.

Définitions

Un graphe orienté G = (V, E) a un ensemble V dont les éléments sont appelés sommets, un ensemble E dont les éléments sont appelés arcs (l'ordre est important).
Pour un arc (x, y) $\in$ E, il est équivalent de dire que x et y sont les extrémités de a, que a est incidente à x et à y, et que y est un successeur de x (et vice versa).
Un graphe est orienté symétrique si $\forall$ arc (x, y), l'arc (y,x) existe.
Un graphe est sans boucle si aucun sommet n'est joint à lui même, c'est-à-dire que $\nexists$ (x, x) $\in$ E.
L'ordre du graphe est le nombre de sommets du graphe.
Un graphe est appelé simple, lorsqu'il ne possède ni boucles ni arêtes parallèles (i.e., une paire de sommets ne peut être jointe que par une arête).

Étude de la connexité

Une chaine est une séquence finie et alternée de sommets et d'arêtes qui commence et finit par des sommets. Les arêtes sont incidentes aux sommets qui les encadrent dans la séquence. Le premier et le dernier sommet sont appelés extrémités de la chaine. Sa longueur est égale au nombre d'arêtes qui la composent.

Un chemin (i.e., chaine orientée) est une séquence finie et alternée de sommets et d'arêtes qui commence et finit par des sommets, chaque arête est sortante d'un sommet et incidente au sommet suivant dans la séquence.

Un circuit est un chemin dont les extrémités coïncident et s'il est en plus élémentaire, on ne rencontre pas deux fois le même sommet.

Soit G=(V, E) un graphe non orienté, G est dit connexe si et seulement si, pour chaque pair de sommets (u, v) $\in$ V, il existe une chaine entre u et v. Soit D=(V, A) un graphe orienté, D est dit fortement connexe si et seulement si, pour chaque pair de sommets (u, v) $\in$ V, il existe un chemin de u à v et un chemin de v à u.

2.3.2 Les réseaux WDM

Les réseau WDM (Wavelength Division Multiplexing) sont des réseaux composés d'un ensemble de nœuds interconnectés entre eux par des fibres optiques. Il sont de trois types. Le premier type est le réseau opaque où le signal est régénéré à chaque nœud par le biais d'un régénérateur. Un nouveau type de réseau est apparu, nommé réseau hybride, utilisant une régénération éparpillée qui est exécutée au niveau des nœuds intermédiaires seulement quand c'est nécessaire.

Ces réseaux évoluent également vers une troisième architecture transparente (i.e., réseaux WDM tout-optiques) où chaque nœud intègre une fonctionnalité de brassage optique du signal, ceci impose la contrainte de continuité de la longueur d'onde qui est le cas étudié ici. Ces réseaux doivent faire face aux difficultés techniques pour vaincre les affaiblissements de la transmission introduits par les composants optiques.

La technologie WDM met en œuvre un multiplexage de longueurs d'onde. L'idée est d'injecter simultanément dans une fibre optique plusieurs signaux numériques sur des longueurs d'onde distinctes.

Cette technique s'avère une solution efficace pour la meilleure exploitation de l'immense bande passante d'une fibre optique. Cette bande est subdivisée en plusieurs canaux travaillant chacun sur une longueur d'onde différente et à un débit adapté à la vitesse de traitement des composants électroniques. L'intégration de cette technique a contribué à l'évolution de la notion de routage qui ne consiste plus à déterminer une route reliant un nœud source à un nœud destination mais aussi une longueur d'onde sur laquelle le trafic sera acheminé [7] .

2.3.3 Reconfiguration

Dans un réseau optique WDM reconfigurable, il est possible de changer la configuration du réseau, c'est-à-dire de modifier le routage de certaines requêtes, soit suite à des évolutions de la matrice de trafic ou pour interdire

l'utilisation de certaines ressources devant subir une opération de maintenance, tout en déplaçant une connexion (e.g., canal de communication optique entre 2 nœuds du réseau, généralement à plus de 2.5 Gbit/s) vers une nouvelle route [1].

L'évolution du trafic au cours du temps, représenté par l'ajout et/ou le retrait des connexions, peut entrainer une mauvaise utilisation des ressources du réseau. La reconfiguration du réseau devient donc nécessaire. Elle consiste à déplacer des connexions sur d'autres routes pour retrouver une utilisation optimale des ressources. Pour celà, on doit non seulement déterminer la nouvelle configuration mais aussi planifier l'ordre dans lequel sont effectués les changements pour passer de la première configuration à la nouvelle, tout en limitant les risques engendrés(e.g., arrêt du trafic ou perte d'informations). Le problème de reconfiguration que nous considérons traite uniquement le déplacement des requêtes (les routages initiaux et finaux sont donnés). Des algorithmes de reconfiguration sont développés dans ce contexte, minimisant chacun un des paramètres telque le nombre de déplacements et les services interrompus.

Ce problème se retrouve dans tous les types de réseaux orientés connexions et il est NP-difficile [10]. Pour le résoudre, nous suivons l'approche de [2]

Lors de la reconfiguration, les dépendances entre la route finale d'une demande et toutes les routes initiales des autres demandes sont modélisées par un graphe orienté, appelé *graphe de dépendance*.
Un graphe de dépendance est définit dans [2] comme suit :

Définition 1. Graphe de dépendance
Étant donné un routage initial R1 et un routage de destination R2,le graphe de dépendance est composé d'un sommet par requête reroutée, et il y a un arc d'un sommet u à un sommet v si la route empruntée par la requête u dans R2 utilise des ressources utilisées par v dans R1.

Exemple : Minimisation des interruptions, modélisation à l'aide d'un jeu d'agents sur le graphe de dépendance

Soit un réseau WDM modélisé par un graphe en grille doté de neuf nœuds et douze liens orientés symétriques. Le nombre des longueurs d'onde sur chaque arc est égale à 1. Soit un trafic de cinq demandes de connexions a, b, c, d, e présentées dans le Tableau 2.1.

La figure 2.5, présente un exemple de routage R1 et R2 des demandes. Si on se place dans le premier cas ou les ressources sont illimitées, on n'aura pas des contraintes lors du passage de R1 à R2. Par contre, si on suppose que les longueurs d'onde sont limitées, nous devons lors du passage de R1 à R2 s'assurer que la longueur d'onde que nous allons utiliser est libre.

TABLE 2.1 – Caractéristiques du réseau européen Atlanta

Demande	Source	Destination
a	4	5
b	1	5
c	2	3
d	1	3
e	6	5

Nous obtenant le graphe de dépendance illustré dans la figure 2.6. A partir de ce dernier, les demandes doivent être traitées dans l'ordre suivant ; b avant a, c avant b, d avant c, a b et c avant d.

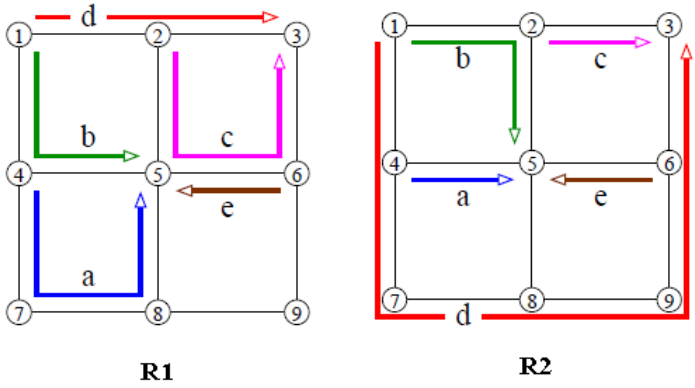

FIGURE 2.5 – Les deux routages R1 et R2 sur la grille orientée symétrique[2]

Le graphe de dépendance généré contient des cycles. Il sera nécessaire dans ce cas d'interrompre temporairement certaines requêtes. Le fait d'interrompre une requête permet d'utiliser ses ressources pour le reroutage d'autres requêtes qui vont eux même libérer des ressources que cette demande a besoin pour être rerouter, citons l'exemple des requêtes b et d. Une interruption est modélisée par le placement d'un agent sur le sommet correspondant à la requête interrompue. Ceci est similaire au jeu des gendarmes et du voleur, où l'objectif est de minimiser soit le nombre total de sommets visités par un agent au cours de la stratégie (i.e., nombre total d'interruptions), soit le nombre d'agents nécessaires à l'existence d'une stratégie (i.e., nombre simultané d'interruptions).

Lorsqu'on place un agent sur un sommet du graphe ceci est équivaut à interrompre la requête correspondante. Tandisque le fait de le retirer d'un

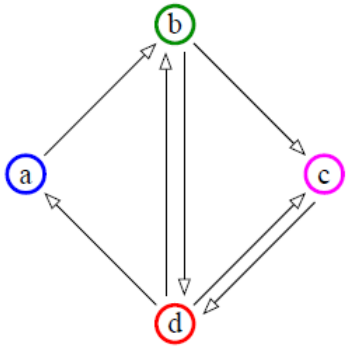

FIGURE 2.6 – Graphe de dépendance de R1 et R2 [2]

sommet du graphe modélise le fait que la requête correspondante est maintenant routée sur sa route finale [1].
Chaque sommet du graphe de dépendance ne peut être traité que lorsque les ressources nécessaires à l'établissement de la nouvelle route de la requête correspondante sont disponibles.

Reprenons l'exemple du graphe de dépendance obtenu à partir de R1 et R2. Tant qu'il contient un cycle, on a intérêt d'interrompre la requête d afin de supprimer le cycle du graphe et traiter les demandes (voir Figure 2.7). Nous avons placé un agent sur le sommet d, cette requête est interrompue temporairement, nous traitons les requêtes c, b et a. Ensuite nous retirons l'agent de d car les ressources dont elle a besoin sont disponibles et elle peut être traitée.

Pour finir, nous décrivons d'autres approches de la littérature pour traiter le problème de reconfiguration du routage.

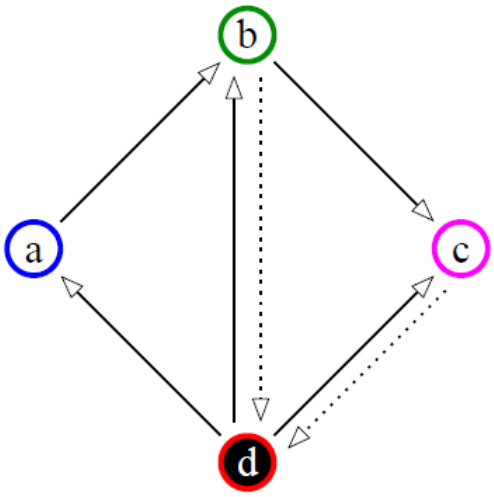

FIGURE 2.7 – Graphe sans cycle

Move-To-Vacant Wavelength-Retuning, MTV-WR

Plusieurs algorithmes ont été développé dans le contexte de la reconfiguration dont la fonction objectif est de minimiser la période durant laquelle le trafic est perturbé. Parmi ces algorithmes, on cite l'approche "Move to vacant wavelength retuning" (MTV-WR). Son principe est de déplacer les requêtes de manière gloutonne jusqu'à atteindre un état satisfaisant. Elle cherche à calculer une nouvelle topologie logique qui diffère de la topologie actuelle par un nombre réduit de demandes à rerouter. Elle garde les routes initiales pour certaines requêtes et il suffit de leur affecter des nouvelles longueurs d'onde. Cette approche se caractérise par une période de perturbation du réseau très courte sans aucune perte de données. Le problème se pose lorsqu'il s'agit d'un réseau dont les nœuds sont équipés de convertisseurs de longueur d'onde, dans ce cas cette approche devient sans intérêt [3] [5].

Least Virtual Hop First, LVHF

LVHF vise à minimiser les longueurs des connexions sur les chemins fixes alternés. Comme pour la phase du reroutage, l'opération du reroutage est restreinte à une seule longueur d'onde pour chaque nouvelle demande de connexion. Cette restriction réduit non seulement la complexité de l'algorithme de reroutage mais aussi la quantité de trafic affecté par l'opération du reroutage. L'opération de reroutage est aussi basée sur l'ensemble des k plus courts chemins calculés pour chaque paire source-destination. Une telle

technique ne prend pas en considération l'état courant du réseau ce qui réduit la complexité de l'algorithme du reroutage [3][4].

Chapitre 3

Ressources illimitées

3.1 Introduction

Dans ce chapitre, nous traitons le problème de reconfiguration du routage lorsque le nombre de longueurs d'onde sur chaque lien n'est pas limité. C'est-à-dire qu'il n'est jamais nécessaire d'interrompre une requêtes.

Nous avons commencé par étudier la complexité du problème et présenter des heuristiques pour le cas d'un réseau particulier, anneau orienté symétrique.

Afin d'évaluer et mettre en valeur les performances des algorithmes mis en œuvre nous avons effectué certaines simulations numériques au niveau de la section 3.5 et nous avons discuté les résultats obtenus.

3.2 Résultats de complexité

Nous considérons le problème dans le cas d'une topologie très simple et montrons la difficulté de ce même problème dans ce cas. Le réseau G, présenté par la Figure 3.1, est composé de 2 nœuds s et t, et de M arcs $a_1, a_2, \cdots, a_M$ de s à t, chaque arc dispose de W longueurs d'onde $\lambda_1, \lambda_2, \cdots \lambda_W$. On considère Q demandes $d_1, d_2, \cdots, d_Q$ chacune de s à t. Une demande est routée à travers un arc du réseau et une longueur d'onde lui est assignée.

Étant donné un routage initial et un routage final des Q demandes (pour chaque requête, un arc + une longueur d'onde), le problème de reconfiguration à 2 nœuds consiste à trouver une séquence de reroutage minimisant la fonction de coût. : $\sum_{e \epsilon R_f} (C(e))^{\alpha}$, pour le cas où α=0.

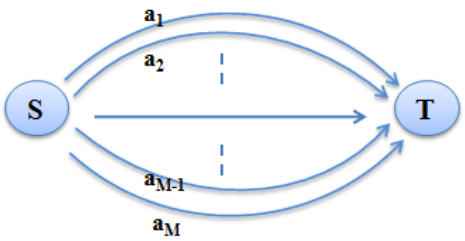

FIGURE 3.1 – Réseau à deux sommets

Nous modélisons une instance du problème de reconfiguration à 2 nœuds par un graphe dirigé D=$(V, A \cup L)$ appelé digraphe de reconfiguration. V, A et L représentent respectivement l'ensembles des nœuds, l'ensemble des arcs et l'ensemble des boucles dans le graphe D considéré. Les nœuds de D représentent les arcs du réseau physique G. À chaque demande d_i qui doit être reroutée, nous associons un arc dans D de u à v si et seulement si la route initiale de d_i correspond au sommet u et sa route finale correspond au sommet v. Nous supposons ici que si une demande utilise le même arc dans le routage initial et le routage final, alors elle ne change pas de longueur d'onde. Pour de telles demandes, nous ajoutons des boucles aux nœuds de D qui correspondent aux arcs concernés dans le réseau.
Soit l'exemple présenté au niveau de la Figure 3.2. Nous avons 3 arcs de s vers t. Chaque arc possède deux longueurs d'onde (λ_u, λ_v), trois demandes a, b et c qui sont initialement routées sur les arcs 1, 2 et 3 et doivent toutes être reroutées sur le lien 2.

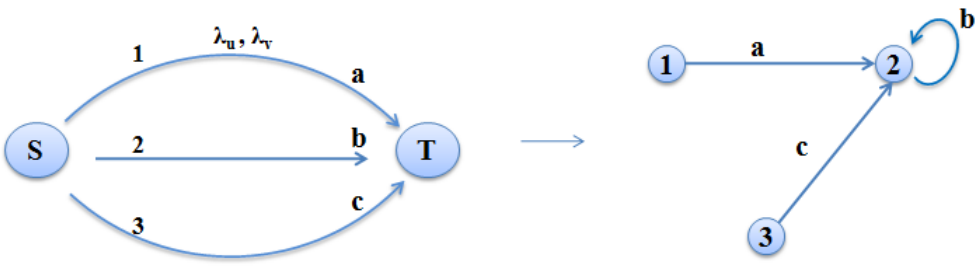

FIGURE 3.2 – Digraphe de reconfiguration

Le Théorème 1 montre que tout digraphe est le digraphe de reconfiguration d'une instance du problème de reconfiguration à 2 nœuds.

Théorème 1. Chaque digraphe D correspond à une instance du problème de reconfiguration dont le réseau physique a 2 nœuds.

Démonstration. Étant donné un digraphe = (V, A ∪ L), nous construisons un réseau composé de 2 nœuds s et t, et composé de $|V|$ arcs. A chaque arc (u, v) ∈ A, nous associons une requête routée à travers l'arc u dans le routage initial et à travers l'arc v dans le routage final. Remarquant que le choix de la longueur d'onde n'est pas important. Finalement à chaque boucle locale (u, u) ∈ L, nous associons une requête routée à travers l'arc u dans le routage initial et final avec la même longueur d'onde. □

Nous définissons une reconfiguration valide R des requêtes par une séquence d'opérations $r_1, r_2, \cdots, r_{|A|}$ sur $D_1, D_2, \cdots, D_{|A|}$. Ces opérations représentent le reroutage séquentiel des $|A|$ requêtes. Le digraphe D_t, $t = 1, \cdots, |A|$, représente le digraphe de reconfiguration après le reroutage de t-1 requêtes. Nous commençons avec D_1 = D. L'opération r_t, t = $1, \cdots, |A|$, sur D_t, est défini comme suit :

- un arc (u, v) ∈ A_t est choisit ;
- nous calculons coût(r_t)=coût((u, v)) ;
- nous construisons $D_{t+1} = (V, A_{t+1} \cup L_{t+1})$ avec $A_{t+1} = A_t \setminus (u, v)$, et $L_{t+1} = L_t \cup (u, v)$.

Remarquons qu'à la fin d'une telle séquence, $D_{|A|+1}$ contient seulement des boucles locales.

Soit D=(V, A ∪ L) un graphe de reconfiguration.

Lemme 2. Étant donné un nœud $v \in V$ avec un degré entrant $d^-(v) \geq 1$, le reroutage des arcs entrants de v coûte un total de $d^-(v)$ ou $d^-(v)$-1.

Démonstration. En premier, remarquons que le reroutage d'une requête, qui correspond à un arc dans D, coûte soit 0 soit 1 parce que $\alpha = 0$. En outre lorsque nous reroutons le première arc entrant de nœud v, une boucle locale est créée au niveau du nœud v, et donc le reroutage des $d^-(v)$-1 autres demandes coûte 1. Donc le reroutage sur les arcs entrants de v coûte un total de $d^-(v)$-1 si le coût de la première requête reroutée est 0 et $d^-(v)$ sinon. □

Dans ce cas, il est intéressant d'étudier le coût d'une reconfiguration tout en considérant le coût sur chaque nœud au lieu du coût sur chaque arc. En effet le Lemme 2 montre qu'une reconfiguration qui minimise le coût total est une reconfiguration qui minimise le nombre des nœuds dont le coût est égale au degrés entrants. Remarquons qu'un nœud v ∈ V avec $d^-(v) = 0$ coûte 0 pour toute reconfiguration.

Lemme 3. Etant donné un graphe acyclique orienté (DAG) D = (V, A), la reconfiguration est simple et le coût de chaque nœud est $d^-(v) - 1$ si $d^-(v) \geq 1$, 0 autrement.

Démonstration. Nous commençons par rerouter tous les arcs entrants d'une feuille de D. Nous le faisons séquentiellement pour le digraphe induit par les nœuds sans boucles. À la fin, le coût de chaque nœud v ∈ V est $d^-(v) - 1$ si $d^-(v) \geq 1$ ou 0 sinon.

Le Lemme 3 montre que la reconfiguration d'un DAG est optimale puisque le coût de chaque nœud v ∈ V est $d^-(v) - 1$ si $d^-(v) \geq 1$ ou 0 sinon. □

Théorème 4. Le problème de reconfiguration à 2 nœuds pour $\alpha = 0$ est NP-difficile et APX-difficile.

Démonstration. Par les précédents Lemmes, le problème de trouver une reconfiguration optimale pour un digraphe D = (V, A∪ L) revient à trouver un plus grand sous-graphe acyclique de D. En effet ceci implique que chaque nœud d'un tel sous-graphe coûte $d^-(v) \geq 1$ ou 0. Ce problème est équivalent au problème des "Minimum Feedback Vertex Set" (MFVS) de D, qui représente un sous-ensemble de nœuds avec cardinalité minimum tel que la suppression de ces nœuds rend D acyclique [6]. Ce problème est NP-difficile et APX-difficile [10]. □

3.3 Études théorique : anneau orienté symétrique

Au niveau de cette section, nous présentons une étude d'un réseau en anneau orienté symétrique. Dans ce cas, nous proposons un algorithme simple pour obtenir une reconfiguration du routage optimale.

Problème

Soient :
Un réseau G = (V, E), V : ensemble des sommets et E : ensemble des arcs.

Une route est l'ensemble des arcs empruntés par une demande de la source vers la destination

Les liens sont de capacité infinie. Pour chaque demande, un routage initial R1 (courant) et un routage final R2 (un nouveau routage) sont donnés.

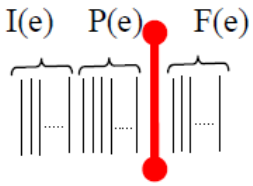

FIGURE 3.3 – La charge sur chaque arête

Avec :

- P(e) : présente le nombre des requêtes permanentes sur e. C'est-à-dire le nombre de requêtes dont le routage initial et final passent par e.
- I(e) : présente le nombre des requêtes initiales sur e. C'est sont le requêtes et qui ne sont pas permanentes, dont le routage initial passe par e mais pas le routage final.
- F(e) : présente le nombre des requêtes finales sur e et qui ne sont pas permanentes, dont le routage final passe par e mais pas le routage initial.

Soit la fonction suivante qui représente le coût total sur l'arête :

$$\sum_{i=I(e)+P(e)}^{I(e)+P(e)+F(e)-1} i^{\alpha} \tag{3.1}$$

L'équation 3.1 peut être comprise entre deux bornes inférieure et supérieure. La borne inférieure est obtenue tout en supposant que I(e)=0, ou que toutes les demandes I(e) sont déplacées de l'arête e avant d'en placer toute autre demande de F(e).
Contrairement à la borne inférieure, la borne supérieure du coût est obtenue tout en plaçant les nouvelles demandes F(e) sur l'arête e avant de déplacer

les demandes initiales I(e).
Donc le coût total de la stratégie sur l'arête est borné comme suit :

$$\sum_{i=P(e)}^{P(e)+F(e)-1} i^{\alpha} \leqslant cout\ total\ sur\ e \leqslant \sum_{i=P(e)+I(e)}^{P(e)+I(e)+F(e)-1} i^{\alpha} \tag{3.2}$$

Définition 2. Graphe de Conflit
Étant donné un routage initial R1 et un routage final R2, le graphe de conflit est composé d'un sommet par requête reroutée, et il y a un arc d'un sommet b vers le sommet a si la route empruntée par la requête b dans R2 occupe toute ou une partie de la route empruntée par a dans R1. Ici on ne prend pas en compte les longueurs d'onde.

Lemme 5. Soit n demandes à rerouter, si on se limite à changer uniquement les longueurs d'onde pour toute demande tout en gardant la même route initiale, on aura une fonction de coût qui est la même pour n'importe quel ordre de traitement. $F = \sum_{e \in E} \sum_{k=1}^{Y_e - 1} k^{\alpha}$

Démonstration. Soient :

- R_I : route initiale pour une demande
- R_F : route finale pour une demande
- $f(d_1, \{\})$: le coût de traiter la demande d_1 sachant qu'aucune autre demande n'a était traitée (graphe initialement vide).
  $f(d_1, \{\})$=0 étant donné que le graphe est initialement vide.
- $f(d_n, \{d_1, ..., d_{n-1}\})$: le coût de traiter la demande d_n sachant qu'on a déjà traité les demandes de d_1 à d_n.
  $\sum_{e \in E} (X_e^{d_n} X_e^{d_1} + X_e^{d_n} X_e^{d_2} + \ldots + X_e^{d_n} X_e^{d_{n-1}})^{\alpha}$
- $\theta = d_1, d_2, ..., d_n$: ordre de traitement quelconque pour les n demandes
- la fonction de coût $f(d_i) = \sum_{e \in R_f} C(e)^{\alpha}$
- Soit la variable binaire $X_e^{d_i}$ qui s'écrit sous la forme suivante :

$$X_e^{d_i} = \begin{cases} 1 & si\ \ e\ \ \in\ \ R_F[d_i] \\ 0 & sinon \end{cases}$$

- Pour vérifier par exemple si deux demandes d1 et d2 passent par une arête commune e, il suffit de faire une simple multiplication de deux variables $X_e^{d_1}$ et $X_e^{d_2}$:

. Si $X_e^{d_1} X_e^{d_2} = 1$: d_1 et d_2 passent toutes les deux par e.

. Si $X_e^{d_1} X_e^{d_2} = 0$: e n'est pas un arc commun à d_1 et d_2.

- Y_e est le nombre de routes finales qui passent par l'arête e.

La fonction de coût pour un ordre de traitement quelconque à n demandes est la suivante :

$$\begin{aligned} F(\theta) &= \sum_{i=1}^{n} f(d_i) \\ &= f(d_1, \{\}) + f(d_2, \{d_1\}) + f(d_3, \{d_1, d_2\}) + \ldots + f(d_n, \{d_1, \ldots, d_{n-1}\}) \\ &= 0 + \sum_{e \in E} (X_e^{d_2} X_e^{d_1})^\alpha + \sum_{e \in E} (X_e^{d_3} X_e^{d_1} + X_e^{d_3} X_e^{d_2})^\alpha + \ldots \\ &\quad + \sum_{e \in E} (X_e^{d_n} X_e^{d_1} + X_e^{d_n} X_e^{d_2} + \ldots + X_e^{d_n} X_e^{d_{n-1}})^\alpha \\ &= \sum_{e \in E} (X_e^{d_2} X_e^{d_1})^\alpha + (X_e^{d_3} X_e^{d_1} + X_e^{d_3} X_e^{d_2})^\alpha \\ &\quad + (\sum_{i=0}^{n-1} X_e^{d_n} X_e^{d_1})^\alpha \end{aligned} \tag{3.3}$$

Donc :

$$F(\theta) = \sum_{e \in E} \sum_{k=1}^{Y_e - 1} k^\alpha \tag{3.4}$$

De ce fait, d'après l'équation 3.3, dans le cas ou le routage initiale n'est pas prit en considération lors du reroutage des demandes, n'importe quel ordre de traitement donnera toujours le même coût. Ces résultats peuvent être assimilés par des valeurs nulles de I(e) ou de F(e).

Avec :

- I(e)=0 : il n'y a pas de demandes qui ont été initialement sur e et qui vont partir par la suite. Donc la charge sur chaque arête augmentera de un à chaque itération. C'est le cas de la borne inférieure de l'equation 3.2.

- F(e)=0 : il n'y a pas de nouvelles demandes qui vont être routées sur cette arête. Donc la charge sur chaque arête diminue de un à chaque itération.

□

Remarque : Si on considère que le graphe sur lequel s'applique le nouveau routage(i.e., routage final) est initialement vide, c'est-à-dire ne pas considérer les routes initiales pour des demandes, on aura de même une fonction de coût constante quelque soit l'ordre de traitement. **Cas d'anneau orienté symétrique**

A ce niveau, nous utilisons les différentes équations déterminée précédemment, au niveau de la section 3.3, pour l'étude du réseau anneau orienté symétrique.

On distingue deux situations extrêmes selon la valeur de α :

- Si $\alpha = 1$; la fonction de coût est égale à la somme des charges sur chaque arête de la route finale d'une demande.

- Si $\alpha = 0$: la fonction de coût sera égale au nombre d'arêtes dans la route finale d'une demande. Dans ce cas le coût sera complètement indépendant de l'ordre de traitement des demandes.

Soit deux ordres de traitement quelconques pour lesquels on ne change que l'ordre de traitement de deux requêtes consécutives a et b.

- C_a^1= Coût du traitement de a après avoir traiter n1 demandes et avant de traiter b.

- C_b^1= Coût du traitement de b après avoir traiter n1 demandes ainsi que la demande a.

- C_b^1= Coût du traitement de b après avoir traiter n1 demandes et avant de traiter a.

- C_b^1= Coût du traitement de a après avoir traiter n1 demandes ainsi que la demande b.

- R_a^i : la route initiale de la demande a.

- R_b^i : la route initiale de la demande b.

- $C^\alpha(a, R/b, e)$: c'est la charge sur l'arête e, s'il fait partie de la route finale de a privée de b, lors de reroutage de a.

- $C^\alpha(b, R/a, e)$: c'est la charge sur l'arête e, s'il fait partie de la route finale de b privée de a, lors de reroutage de b.

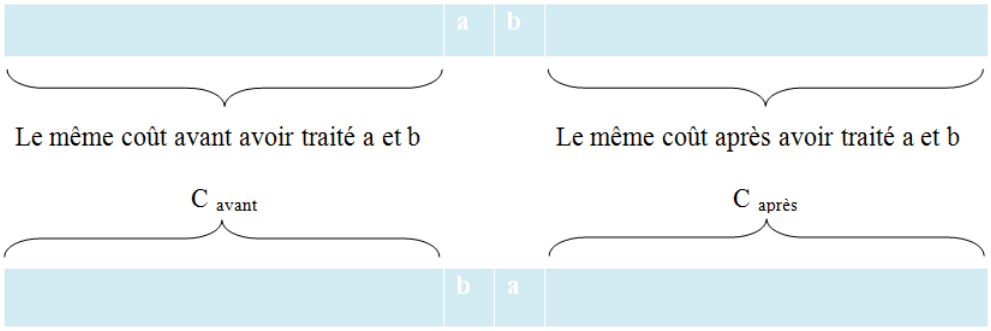

FIGURE 3.4 – les deux ordres de traitement

Le coût de traitement des n1 demandes avant de considérer a et b est le même pour les deux ordres considérés. De même le coût de traitement de toutes les autres demandes qui suivent a et b (n2 demandes) sera le même étant donné que dans les deux cas a et b seront déjà routées (voir Figure 3.4). Donc les coûts du premier et du deuxième ordre de traitement peuvent être modélisé de la façon suivante :

$$\begin{array}{rcl} Cout1 & = & C_{avant} + C_a^1 + C_b^1 + C_{apres} \\ Cout2 & = & C_{avant} + C_b^2 + C_a^2 + C_{apres} \end{array} \tag{3.5}$$

Pour router les deux demandes a et b, on a 2 cas, le premier s'intéresse aux demandes qui ont même sens de rotation sur l'anneau, tandis-que le deuxième cas traite les demandes dont les sens de rotation sont opposés. Pour ces deux cas, on va déterminer la fonction de coût pour le traitement de a en premier lieu ensuite b et inversement.
Nous avons considéré une capacité infinie pour tous les lemmes qui suivent, c'est-à-dire qu'il n'y a pas de contrainte sur le nombre de longueur d'onde, les ressources sont toujours disponibles.

Lemme 6. Soit une valeur de $\alpha = 1$. Pour deux demandes consécutives et de même sens de rotation sur un réseau en anneau, le fait d'inverser leur ordre de traitement donnera le même coût.

Démonstration. Soient deux demandes successives a et b de même sens de rotation sur l'anneau. Le fait de changer leurs ordres de traitement, modifie la valeur cout. Les deux cas qui se présentent sont comme suit :

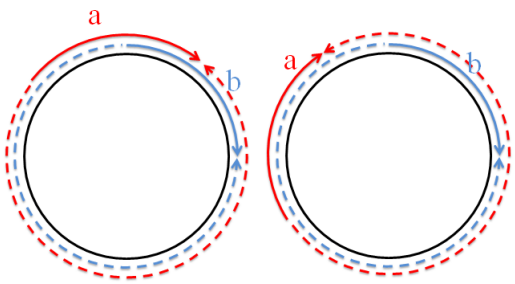

FIGURE 3.5 – a et b ont même sens de rotation

- Traiter a avant b :

$$\begin{array}{rcl} C_a^1 & = & \sum_{e \in E} C^\alpha(a, R/b, e) \\ C_b^1 & = & \sum_{e \in R_a^i \cap R_b^i} C^\alpha(b, R/a, e) + \sum_{e \in E/(R_a^i \cap R_b^i)} (C(b, R/(a,b), e) + 1)^\alpha \end{array}$$

- Traiter b avant a :

$$C_b^2 = \sum_{e \in E} C^\alpha(b, R/a, e)$$
$$C_a^2 = \sum_{e \in R_a^i \cap R_b^i} C^\alpha(a, R/b, e) + \sum_{e \in E/(R_a^i \cap R_b^i)} (C(a, R/(a,b), e) + 1)^\alpha$$

Selon la valeur de α, on aura :

- Si $\alpha = 1$: les deux stratégies 1 et 2 ont le même coût

- Si $\alpha \neq 1$: on aura différents coûts.

□

Lemme 7. Si on suppose que toutes les demandes ont la même longueur (Nombre d'arêtes par lesquelles passent chaque requête de la source à la destination) et pour une valeur de $\alpha = 1$, n'importe quel ordre choisit donnera nécessairement le même coût.

Remarque : On peut avoir le même coût tout en inversant l'ordre de traitement de deux demandes qui ne sont pas forcement consécutives à condition que le coût intermédiaire soit fixe.

Lemme 8. Si $\alpha = 1$ et si les demandes ont différentes longueurs et sens opposées, il faut commencer par traiter les demandes dont les longueurs sont plus élevées afin d'avoir une fonction de coût minimale.

Démonstration. Considérons l'exemple de deux demandes a et b, présenté au niveau de la Figure 3.6.
Les demandes a et b possèdent respectivement les longueurs 2 et 3 arêtes. Deux cas de traitement sont possibles :

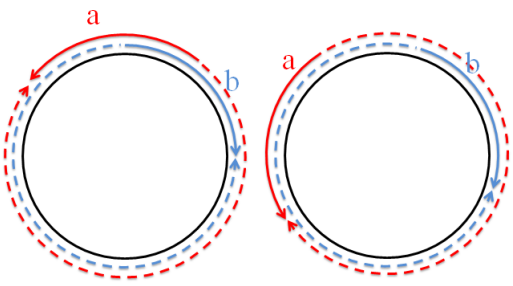

FIGURE 3.6 – a et b ont sens opposés de rotation

Cas 1 : ab

$$\begin{aligned} C_a^1 &= \sum_{e\in E/(R_b^i\ (R_b^i\cap R_b^i))} C^\alpha(a,R/b,e) + \sum_{e\in(R_b^i\ (R_b^i\cap R_b^i))}(C(a,R/(a,b),e)+1)^\alpha \\ C_b^1 &= \sum_{e\in E} C^\alpha(b,R/a,e) \end{aligned}$$

Cas 2 : ba

$$\begin{aligned} C_b^2 &= \sum_{e\in E/(R_a^i\ (R_b^i\cap R_b^i))} C^\alpha(b,R/a,e) + \sum_{e\in(R_a^i\ (R_b^i\cap R_b^i))}(C(b,R/(a,b),e)+1)^\alpha \\ C_a^2 &= \sum_{e\in E} C^\alpha(a,R/b,e) \end{aligned}$$

La différence entre les 2 cas réside dans les deux sommes :

$$\sum_{e\in(R_b^i\ (R_b^i\cap R_b^i))} (C(a,R/(a,b),e)+1)^\alpha \tag{3.6}$$

$$\sum_{e\in(R_a^i\ (R_b^i\cap R_b^i))} (C(b,R/(a,b),e)+1)^\alpha \tag{3.7}$$

Tant que a est plus petite que b, alors le coût apporté par le traitement de b ensuite a est mois élevé. Donc il faut mieux commencer par traiter les demandes dont les longueurs sont plus grandes. □

Théorème 9. Soient :
Un réseau en anneau G = (V, E), n demandes $(r_1, r_2, ..., r_n)$, $\alpha = 1$.
$|r_i|$: représente la longueur de la demande i. Si $|r_1| \geq |r_2| \geq |r_3| \geq ... \geq |r_n|$, alors l'ordre optimal est : $\theta = (r_1, r_2, ..., r_n)$.

Démonstration. Soit $|r_i|$: longueur de la i_{me} demande
Montrons que $\theta = (r_1, r_2, ..., r_n)$, avec $r_1 \geq r_2 \geq ... \geq r_n$ est un ordre optimal.
Soit $\theta' = (r_1', r_2', ..., r_n')$ un ordre quelconque.
Montrons que s'il existe $j < n$ tel que $|r_j'| \leqslant |r_{j+1}'|$, alors $\theta'' = (r_1', r_2', ..., r_{j+1}', r_j', ..., r_n')$ a un coût au plus égal au coût de θ'.

- Si $|r_j'| = |r_{j+1}'|$, d'après Lemme 6(si les requêtes ont le même sens) et d'après Lemme 8(si les requêtes ont le sens opposés), θ' et θ'' ont même coût.

- Si $|r_j'|$ et $|r_j''|$ ont même sens, d'après le Lemme 6, θ' et θ'' ont même coût.
  En effet :
  coût(θ'')=coût(θ')
  $=f(d_1,\{\})+f(d_2,\{d_1\})+\cdots+f(d_{j+1},\{d_1,\cdots,d_{j-1}\})+f(d_j,\{d_1,\cdots,d_{j-1},d_{j+1}\})\ldots+$
  $f(d_n,\{d_1,\ldots,d_{j-1},d_{j+1},d_j,\cdots,d_{n-1}\})$

- Supposons maintenant que $|r_j^{'}|$ et $|r_{j+1}^{'}|$ ont deux sens opposés, alors, d'après le lemme 8, coût $(\theta^{''})$<coût $(\theta^{'})$.

Pour conclure, si $\theta^{'}$ est optimal, on peut par une suite d'inversions des requêtes deux à deux, obtenir l'ordre θ, sans augmenter le coût.
Donc θ est optimal.

□

Lemme 10. Soit un réseau quelconque G(V, E) et un graphe de conflit sous forme d'un DAG (Direct Acyclic Graph). Si on commence par traiter les demandes tout en partant des feuilles du graphe de conflit, alors tout ordre compatible donne le même résultat optimal.

Démonstration. L'expression du coût total sur une arête e est le suivant :

$$\sum_{i=I(e)+P(e)}^{I(e)+P(e)+F(e)-1} i^{\alpha}$$

Dans le cas ou on commence par traiter les demandes qui existent au niveau des feuilles du graphe de conflit, la variable I(e) sera nulle sur toutes arêtes qui appartiennent aux routes finales des demandes. Donc l'expression du coût total sur ces arêtes sera :

$$\sum_{i=P(e)}^{P(e)+F(e)-1} i^{\alpha}$$

Pour le cas ou $\alpha = 1$, l'expression du coût total sur chaque arête sera suivante :

$$\begin{aligned} & \textstyle\sum_{i=P(e)}^{P(e)+F(e)-1} i \\ = \; & \textstyle\sum_{i=0}^{F(e)-1} P(e) + i \\ = \; & F(e) \times P(e) + \textstyle\sum_{i=0}^{F(e)-1} i \end{aligned}$$

Donc, on aura toujours le même coût quelque soit l'ordre de traitement(voir calcul au niveau de l'equation 1.2). □

3.4 Présentation des algorithmes

Pour étudier le problème de reconfiguration du routage, nous avons développé certaines heuristiques. Nous présentons à ce niveau celles qui traitent le cas des ressources illimitées, pour les autres approches, elles seront détaillées par la suite au niveau de la section 4.3.

Pour l'ensemble des algorithmes développés, une variable globale nommée "List" permet de déterminer pour chaque demande i sa route initiale et sa route finale. Une route est un ensemble d'arêtes dans le graphe physique, que nous dénotons par R_I et R_F pour la route initiale et la route finale respectivement. Puisque le nombre de longueurs d'onde est limité, plusieurs requêtes peuvent partager la même arête, elles auront simplement des longueurs d'onde différentes.

3.4.1 Algorithmes du routage

Pour une capacité = ∞, il est suffisant de générer deux routages aléatoires pour pouvoir tester les approches, l'algorithme développé pour un routage aléatoire est le suivant :

Algorithm 1 Routage Aléatoire : $(Graphe_{physique}) \mapsto$ Routage

```
route ← Pile vide
S(i) ← nœud source de la demande i du List
T(i) ← nœud destination de la demande i du List
neighbors(vi) ← voisins de vi // les successeurs du sommet vi dans Graphe_physique
for i=0 to |List|-1 do
    route ← route ∪ S(i)
    while route.peek ≠ T(i) do
        choisir un sommet v ∈ neighbors(route.peek)
        if v ∉ route then
            route ← route ∪ v
        else
            while route.peek ≠ v do
                route ← route - route.peek // supprimer les successeurs de v dans route
        if route=∅ then
            route ← route ∪ S(i)
return route
```

Cet algorithme nous sert dans la suite à générer des instances pour tester nos heuristiques de reconfiguration du routage.

3.4.2 Approche selon la Matrice

Au niveau de cette approche, nous cherchons à déterminer la dépendance entre les demandes qui ne sont pas encore rerouter. Nous utilisons une matrice

à n lignes et n colonnes $M[n][n]$, chaque ligne et chaque colonne représente une demande parmi n.
Chaque case de la matrice est définit comme suit :
$\forall$ e $\in R_F[l]$, M_{kl} = coût de rerouter l après avoir rerouté k - coût de rerouter l maintenant
Ceci revient à déterminer, pour chaque ligne, la dépendance entre la demande désignée par cette ligne et les autres demandes désignées par les colonnes. C'est pour cela, on aura des zéros sur toute la diagonale.
Si la valeur de $M_{kl} \leqslant 0$ alors la demande k passe par un ou plusieurs arcs de la route finale de l, on a donc intérêt à la rerouter avant l pour diminuer le coût de reroutage de l.

Suite au remplissage de la matrice, nous devons choisir la demande à rerouter à cet instant et l'ajouter à l'ordre de traitement. Nous effectuons ce choix selon l'une de deux méthodes, SML et SMLNN.

Somme Minimale des Lignes (SML)

Soit une instance I du problème suivante :

- un réseau G, graphe orienté symétrique
- n demandes à rerouter.
- $R_I[n]$: route initiale de la demande n.
- $R_F[n]$: route finale de la demande n.
- un routage initial $R_I = (R_I[1], R_I[2], \cdots, R_I[n])$
- un routage final $R_F = (R_F[1], R_F[2], \cdots, R_F[n])$

Soient :

- M^j(I,$\{r_1, r_2, \cdots, r_n\}$)=$[M^j_{kl}] \in \mathbb{R}^{(n-(j-1))\times(n-(j-1))}$: La matrice de l'instance I obtenue à un instant j, suite au reroutage de l'ensemble des demandes $r_1, r_2, \cdots, r_n$.
- M^j_{kk}=0
- $\forall$ k $\neq l, M^j_{kl} = cout_1(I, \{r_1, r_2, \cdots, r_{j-1}, d_k\}) - cout_2(I, \{r_1, r_2, \cdots, r_{j-1}\})$
- $M[i][j] = |(R_F[i]/R_I[i]) \cap R_F[j]| - |(R_I[i]/R_F[i]) \cap R_F[j]|$

Le principe de fonctionnement de l'approche SML est le suivant :

- Calculer la matrice M^0
- Pour tout i $\in 1 \cdots$ n faire
  - Choisir une demande, parmi celles qui ne sont pas encore reroutées et qui minimise la somme sur les lignes : $\sum_{l=1}^{n} M_{kl}^{i-1}$
  - Rerouter k
  - Calculer $M^i(I, \{d_k\})$: la i^{eme} matrice de l'instance I après le reroutage de la demande d_k, donc à cet instant i demandes sont reroutées $r_1 \cdots r_{i-1}+d_k$

L'algorithme 2 détermine la fonction de coût de reroutage des m demandes selon un ordre passé en paramètre. Pour une demande donnée, son coût $\sum_{e \in R_f} C(e)^\alpha$ représente la charge sur l'ensemble des arcs sur lesquels passe cette demande dans sa route finale.

Algorithm 2 Coût : (Tableau Ordre [0$\cdots$ m-1], double α) $\mapsto$ double Z

```
1: m_physique ← nombre d'arcs dans le graphe physique
2: m ← nombre des demandes
3: R_F[1···m] ← pour chaque demande sa route finale
4: R_I[1···m] ← pour chaque demande sa route initiale
5: cout ← 0
6: coutf ← 0
7: for i = 0 to m_physique do
8:     cout1← {R_I[e], e ∈ E_demandes|i ∈ R_I[e] } //nombre des routes initiales traversant l'arc
9:     for j = 0 to m-1 do
10:        if Ordre[j]∈ R_F[i] then
11:            coutf ← coutf + cout1^α
12:            cout1++
13:        if Ordre[j]∈ R_I[i] then
14:            cout1 - -
15: return coutf
```

Algorithm 3 Ordre via SML : ($Graphe_{demande}$, $Graphe_{physique}$,double α, fonction Coût(double α, $Graphe_{physique}$) $\mapsto$ double v)) $\mapsto$ (double c, Tableau Ordre[0$\cdots$ m-1])

1: On considère m demandes
2: $M_{Cout} \leftarrow$ Matrice carrée de taille m*m
3: $Rerouted \leftarrow$ Tableau booléen de taille m
4: Ordre $\leftarrow$ Tableau des entiers de taille m
5: **for** $i = 0$ to m-1 **do**
6: $\quad Rerouted[i] \leftarrow$ false
7: fin $\leftarrow$ m
8: **while** fin>0 **do**
9: $\quad$ **for** $i = 0$ to m-1 **do**
10: $\quad\quad$ **for** $j = 0$ to m-1 **do**
11: $\quad\quad\quad$ **if** Rerouted[i]=false **then**
12: $\quad\quad\quad\quad$ **if** i=j **then**
13: $\quad\quad\quad\quad\quad M_{Cout}[i][j] \leftarrow 0$ *// 0 sur la diagonale*
14: $\quad\quad\quad\quad$ **else**
15: $\quad\quad\quad\quad\quad M_{Cout}[i][j] \leftarrow$ Coût Séquence(Rerouted, i, j, α)
16: $\quad T_{sum}[i] \leftarrow \sum_{j=0}^{m-1} M_{Cout}[i][j]$ *// Somme des lignes*
17: $\quad min_{ligne} \leftarrow$ i tel que $T_{sum}[i]=$ min $T_{sum}[i]$
18: $\quad Rerouted[min_{ligne}] \leftarrow$ true *// demande marquée reroutée*
19: $\quad$ Ordre[ind]$\leftarrow min_{ligne}$: Ajouter à l'ordre de traitement des demandes
20: Cost $\leftarrow$ Coût(Ordre, α)
21: **return** $(Cost, Ordre)$

Lors de son exécution, l'algorithme 3 fait appel à une fonction "Coût Séquence". La valeur récupérée permet de remplir une matrice utilisée afin de choisir la quelle des demandes rerouter à cet instant. Ce choix se fait selon la somme effectuée sur chaque ligne de la matrice. Une valeur négative de la somme signifie que cette requête influe sur les coûts de reroutage d'une ou plusieurs demandes.

Algorithm 4 Coût Séquence : (Tableau T [0$\cdots$ m-1], entier i, entier j, double alpha) $\mapsto$ double Z

1: $R_F[1 \cdots m] \leftarrow$ pour chaque demande sa route finale
2: $R_I[1 \cdots m] \leftarrow$ pour chaque demande sa route initiale
3: **return**
$$\sum_{\substack{e \in R_{F[i]} \\ e \in R_{F[j]}, e \notin R_{I[i]}}} (C(e)+1)^\alpha + \sum_{\substack{e \in R_{F[j]} \\ e \in R_{F[i]}}} (C(e)-1)^\alpha + \sum_{\substack{e \in R_{F[j]} \\ e \notin R_{F[i]}, e \notin R_{I[i]}}} C(e)^\alpha - \sum_{e \in R_{F[j]}} C(e)^\alpha$$

Somme Minimale des Lignes à éléments Négatifs ou Nuls (SMLNN)

Nous avons également étudié une approche alternative SMLNN qui choisit une ligne (i.e, une requête) parmi celles dont les éléments sont tous négatifs ou nuls. Le but est de déterminer quelles sont les demandes que lorsqu'on les reroute ça influe sur le coût de reroutage de toutes les autres demandes qui ne sont pas encore reroutées. Son implémentation consiste à ajouter, après la ligne 16 de l'algorithme 3, les deux lignes suivantes :

$T_{negatif}[i] \leftarrow T_{sum}[i]$ tel que $\vee i \in [1 \cdots m] M_{Cout}[i][j] \leqslant 0$
$min_{ligne} \leftarrow i$ tel que $T_{negatif}[i] = minT_{negatif}[i]$

3.4.3 Approche selon le Tri

Cette approche commence par générer un ordre aléatoire, ou prend un ordre généré par l'une des deux premières approches. Ensuite elle parcoure les demandes et vérifie à chaque itération s'il faut rerouter une demande avant la suivante. Si oui, alors l'ordre des deux demandes est inversé. Le test continue pour la demande suivante et celle qui lui succède. Ce traitement s'arrête dès qu'on arrive à faire un parcours complet des demandes sans inversion. Ceci signifie que les demandes sont placées dans le bon ordre ce qui génère un coût performant. L'approche est décrite formellement dans l'Algorithme 5.

Algorithm 5 Ordre via Tri : (Arraylist Ordre, int niveau, double α) $\mapsto$ (double c, Tableau ordre[0$\cdots$ m-1])

```
1: On considère m demandes
2: longueur ← taille de Ordre //nombre des demandes
3: CostOpt ← Coût(Ordre, α)
4: cost ← 0
5: OrdreOpt ← Ordre
6: tab ← Ordre
7: permut ← false // variable utilisée pour indiquer la fin des permutations

8: repeat
9:     permut← false
10:    for i = niveau to longueur do
11:        for j = niveau to longueur do
12:            if i=j and i≠ longueur-1 then
13:                tab[j]← OrdreOpt[j+1]
14:                tab[j+1]← OrdreOpt[j]
15:            cout1 ← Coût(OrdreOpt, α)
16:            cout2 ← Coût(tab, α)
17:            if cout2<cout1 then
18:                permuter(OrdreOpt[j], OrdreOpt[j+1])
19:                CostOpt← cost2
20:                permut ← true
21: until (permut=true)
22: return (CostOpt, OrdreOpt)
```

3.4.4 Approche selon les longueurs des requêtes

Cette approche a été développée dans le but de vérifier si l'idée présentée au niveau du Théorème 9 reste valable quelque soit le réseau et pour différentes valeurs de α. Le principe du Théorème 9 dit que le fait de réordonner les demandes selon un ordre décroissant de la longueur produit un meilleur ordre de traitement des demandes et un meilleur coût.
Suite à la génération des demandes, nous calculons la longueur de chacune d'entre elles, nous les ordonnons tout en commençant par les demandes les plus longues.

Le principe de fonctionnement de cette approche est illustré au niveau de l'algorithme 6.

Algorithm 6 Ordre selon les longueurs : (List demandes) $\mapsto$ Tableau ordre[0$\cdots$ m-1]

```
1: order=∅ ← tableau des entiers
2: index=∅ ← tableau contenant les identifiants des requêtes
3: length=∅ ← tableau contenant les longueurs des requêtes
4: n ← |demandes|
5: for i = 0 to n do
6:       length[i]← |i|//taille de la requête i
7:       index[i]← |i|
8: Trie décroissant de length
9: for j = 0 to n do
10:      ind ← indice de (length[i]) dans index
11:      order[i]← ind
12:      index[ind]← 0
13: return ordre
```

3.5 Simulations

Le but de ces simulations est de comparer les performances des approches proposées. Les résultats présentés sur les courbes sont des moyennes calculées sur 100 simulations.

Différents scénarii ont été considérés. Ils dépendent principalement de la topologie physique adoptée pour la représentation du réseau étudié et également du nombre de requêtes de connexion ainsi que d'autres paramètres en relation avec les algorithmes développés (e.g., nombre d'itérations effectuées...).

Nous commençons par préciser les différents paramètres considérés, puis nous analysons les résultats.

3.5.1 Paramètres de simulations

Lors de la conception des approches de reconfiguration du routage, nous avons fixé certains paramètres. En effet, nous avons considéré la topologie physique du réseau Cost266 dotée de 37 nœuds et 57 liens. Ce réseau, représenté dans la Figure 3.7, est récupéré à partir du SNDlib.

Les différentes matrices de trafic ont été générées d'une manière aléatoire selon une distribution uniforme, tout en respectant la contrainte du nombre des nœuds et des liens disponibles dans le réseau considéré. Ainsi, pour chaque demande de trafic, les nœuds source et destination sont choisis dans l'intervalle [1,37].

Pour tracer les différentes courbes, nous avons généré 100 scénarios de test. Chaque scénario est la génération aléatoire d'une matrice de trafic de 100 requêtes.

FIGURE 3.7 – Topologie de réseau Cost266

3.6 Scénarii de simulations

La Figure 3.8 présente 6 histogrammes. C'est sont les 6 approches développées pour la résolution du problème de reconfiguration du routage dans le cas des ressources illimitées.

Si nous comparons les deux approches de la matrice, SML et SMLNN, nous constatons que les différences ne sont pas significatives. Le SML aboutit parfois à un coût meilleur que celui généré par le SMLNN et le contraire arrive également.

Les approches Tri-SML et Tri-SMLNN implémentent l'approche de Tri, elles reçoivent en paramètres l'ordre généré par la première et la deuxième approche. Cette approche s'avère être une amélioration des deux premières

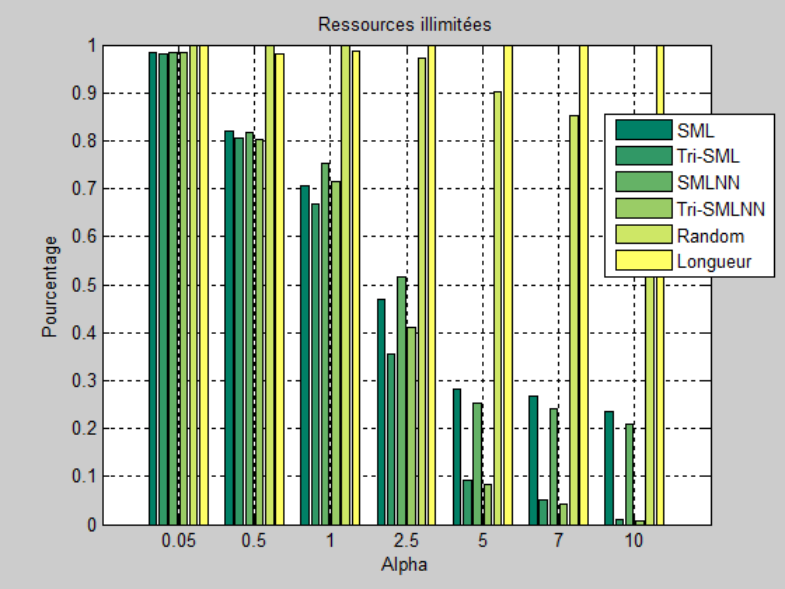

FIGURE 3.8 – Ressources illimitées

approches tant qu'elle prend comme paramètre l'ordre généré par l'une des deux et essaye de réordonner les demandes de façon à aboutir à un coût meilleur. Cependant, elle est parfois incapable de réordonner les demandes même s'il est possible d'avoir un autre ordre et un coût meilleur.

Ce phénomène s'explique par le fait qu'elle traite les demandes deux à deux, donc on peut se trouver avec un cas pour lequel on a intérêt à inverser l'ordre de traitement de deux demandes qui sont éloignées par une demande ou plus. Nous pouvons optimiser les résultats obtenus par cette approche, il suffit de changer le principe du tri utilisé au niveau de l'algorithme 5. Au lieu de tester a chaque deux demandes voisins, on peut augmenter ce nombre à 3 ou plus.

Si on observe les résultats obtenus par l'approche Random, on constate que l'ordre généré aléatoirement produit toujours un coût moins que les quatre premières approches. Mais Ça peut arriver qu'une telle approche génère un ordre et un coût qui est meilleur que les coûts obtenus par les autres approches.

Pour l'approche Longueur, les demandes ont été réordonnées selon un ordre décroissant de la longueur. Cette approche a été développé dans le but de tester si le Théorème 9 est toujours vérifié quelque soit le réseau et pour différentes valeurs de α. Lorsqu'on compare les résultats obtenus par cette approche avec les autres, on se rend compte que le Théorème 9 reste valable

seulement dans le cas d'un réseau orienté symétrique et pour une valeur de α.

Influence de la valeur de α sur le coût de reroutage

La courbe, présentée au niveau de la Figure 3.9 est une représentation des valeurs de coûts obtenus par l'approche SML. Elle illustre la variation du coût de reroutage en fonction des valeurs de α.

On constate que la courbe a une allure croissante. Ceci permet de conclure que pour des valeurs de $\alpha > 1$, rajouter une longueur d'onde sur une fibre très utilisée coûte plus cher que sur une fibre moins utilisée.

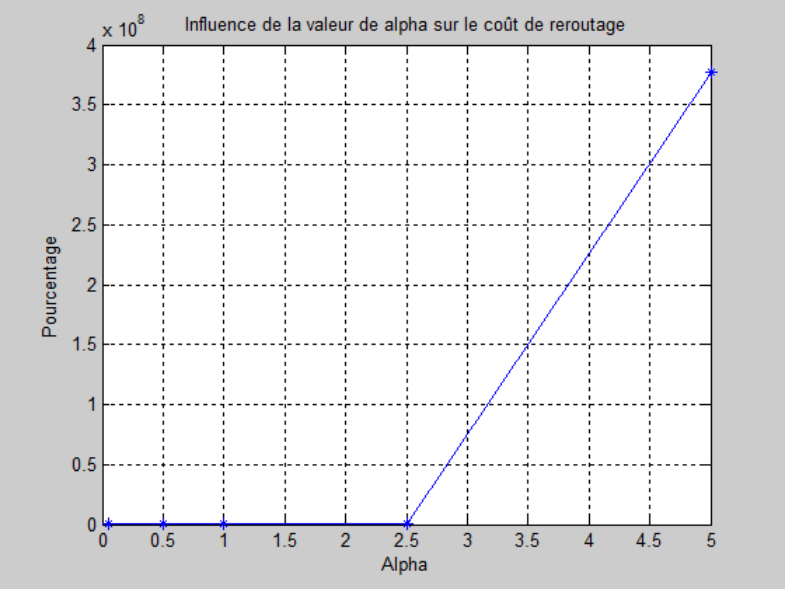

FIGURE 3.9 – Influence de la valeur de α sur le coût de reroutage

La valeur de α constitue un paramètre important à prendre en considération lors du calcul de coût de reroutage. Il faut mieux étudier toujours les valeurs du coût pour toutes les valeurs de α, car ça dépend de la technologie utilisée pour le reroutage.

Chapitre 4

Ressources limitées : Traitement de graphe de dépendance

4.1 Introduction

Nous avons proposé dans le chapitre précédent des approches qui minimisent le coût de reroutage dans le cas des ressources illimitées. Cependant, pour être plus proche de la réalité, il est nécessaire de considérer le nombre des longueurs d'onde disponibles sur chaque arête du graphe. Dans ce chapitre, nous présentons d'abord les approches de traitement des demandes tout en tenant compte de la rareté des ressources. Nous concluons avec des résultats de simulations.

4.2 Méthode pour obtenir un DAG connexe

Suite à la génération de deux routages initial et final, un graphe de dépendance sera généré, présentant la dépendance entre la route finale d'une demande et toutes les routes initiales des autres demandes (voie Section 1). Le graphe de dépendance ainsi généré peut contenir des cycles. Pour pouvoir le traiter et déterminer un ordre de reroutage des requêtes, il faut interrompre temporairement certaines requêtes, ce qui correspond à supprimer les cycles dans le graphe de dépendance.

Plusieurs approches ont été développées dans ce contexte, parmi elles, nous citons le cas des approches qui cherche à minimiser le nombre total de requêtes à interrompre au cours de traitement, ce nombre n'est autre que l'indice de transmission (taille d'un minimum feedback vertex set, MFVS) du graphe de dépendances [6].

En effet, l'indice de transmission est la taille du plus petit ensemble de sommets à supprimer d'un graphe orienté pour le rendre acyclique.
La figure 4.1 présente un cas du graphe contenant des cycles. En appliquant la notion de MFVS sur ce graphe, nous obtenons MFVS=1. Il suffit donc d'interrompre le nœud coloré en rouge qui apparait dans le deuxième graphe de la figure pour avoir un DAG. Une fois la requête rouge est interrompue, les autres requêtes peuvent être traitées (déplacées) séquentiellement en partant des feuilles du graphe de dépendance.

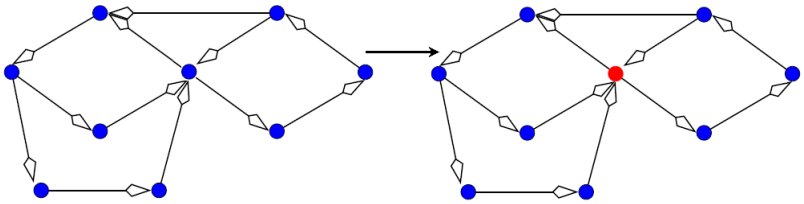

FIGURE 4.1 – Graphe avec cycle [2]

Dans ce même contexte et afin de faciliter le traitement de graphe de dépendance, nous utilisons l'approche développée par David Coudert et Dorian Mazauric de l'équipe Mascotte. Cette approche, NodesForCycle, détermine l'ensemble des sommets à supprimer du graphe pour qu'il soit acyclique. Cependant, le graphe obtenu peut ne pas être connexe. Pour éviter ce cas, nous récupérons l'ensemble des sommets à supprimer, mais nous ne supprimons que leurs arcs entrants. Ainsi, le graphe obtenu est acyclique et connexe.

Si on reprend l'exemple du graphe de la figure 4.1, le fait de supprimer les arcs entrant de sommet coloré en rouge rend le graphe un DAG connexe, on obtient le nouveau graphe illustee au niveau de la figure 4.2.

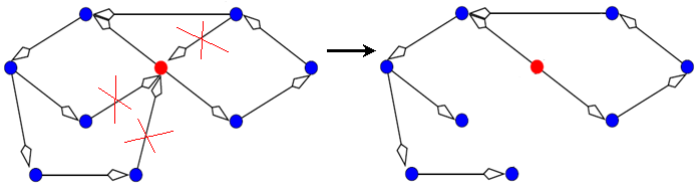

FIGURE 4.2 – Suppression des cycles, MFVS = 1

L'algorithme développé pour la génération du graphe de dépendance est le suivant :

Algorithm 7 Graphe dépendance : ($Graphe_{physique}$ g, $Graphe_{demandes}$) $\mapsto$ $Graphe_{dependance}$

Soit $Graphe_{dependance}$ =(V, E), V=$\emptyset$, E=$\emptyset$ // *graphe physique initialement vide, V est l'ensemble des arêtes et E est l'ensemble des sommets*
memory : v $\in$ V($Graphe_{dependance}$) $\mapsto$ e $\in$ E($Graphe_{demandes}$) //*associe à chaque sommet du graphe de dépendance un arc du graphe de demandes*

DemandI $\leftarrow$ ensemble des demandes dont les routes initiales passent par l'arête e
DemandF $\leftarrow$ ensemble des demandes dont les routes finales passent par l'arête e
$\lambda_i(d_i) \leftarrow$ ensemble des longueurs d'onde utilisées par d_i dans le routage initial
$\lambda_f(d_i) \leftarrow$ ensemble des longueurs d'onde utilisées par d_i dans le routage final
for e $\in$ $Graphe_{demandes}$ **do**
    v $\leftarrow$ créer un nouveau sommet
    $Graphe_{dependance} \leftarrow$ v // *ajouter le sommet au graphe*
    memory $\leftarrow$ (ID(e), v) // *ajouter le doublet "identifiant arc-sommet"*
$\forall$ e $\in$ E
taille1 $\leftarrow |DemandF|$
taille2 $\leftarrow |DemandI|$
for i=0 to taille1 **do**
    $d_i \leftarrow$ List(DemandF (i)) // *List est la variable globale qui contient l'ensemble des requêtes*
    l1 $\leftarrow \lambda_f(d_i)$
    for j=0 to taille 2 **do**
        $d_j \leftarrow$ List(DemandF(j))
        l2 $\leftarrow \lambda_f(d_j)$
        if l1 $\cap$ l2 $\neq \leftarrow$ **then**
            $id_S \leftarrow$ identifiant de d_i
            $id_T \leftarrow$ identifiant de d_j
            $E(Graphe_{dependance}) \leftarrow E(Graphe_{dependance}) \cup E(memory(id_S), memory(id_T))$ //*créer un nouveau arc entre* $(memory(id_S), memory(id_T))$
return $Graphe_{dependance}$

Afin d'enlever les cycles dans le DAG ainsi généré, on procède comme suit :

For s ∈ NodesForCycle(graph) **do**
Supprimer les arêtes entrantes de s

En d'autres termes l'approche présentée dans cette section, permet de décider des requêtes à interrompre pour réaliser le routage. Une fois ces requêtes interrompues, l'ordre de traitement est guidé par le graphe de dépendance, qui est alors un DAG. Pour déterminer dans quel ordre (compatible avec le DAG) les requêtes doivent être traitées pour minimiser le coût, nous considérons diverses heuristiques qui réutilisent celles proposées dans la section précédente.

4.3 Présentation des algorithmes

Considérant le cas d'un graphe de dépendance sous forme d'un DAG connexe, lors du son traitement, on commence du bas en haut ; des feuilles à la racine. Ceci est dut au fait que les ressources dont les demandes ont besoin pour être rerouter sur les nouvelles routes sont disponibles. Deux approches sont développées permettant de traiter le graphe de dépendance différemment. La première porte sur la notion de traitement par feuilles, tandis que la deuxième procède par le traitement par niveau.

4.3.1 Algorithmes du routage

Pour la génération du graphe de dépendance, nous avons besoin de deux routages différents. Nous avons généré le premier routage R1 à l'aide d'un routage Glouton (Algorithme 8), tandis que le deuxième routage R2 est généré à l'aide d'une approche sous forme de programmation linéaire développée par Issam Tahiri de l'équipe Mascotte.

Algorithm 8 Routage Glouton : ($Graphe_{physique}$ g, ListRequêtes) $\mapsto$ nbr-Lambda

1: nbrKPath $\leftarrow$ 1 *//nombre de k plus chemins*
2: *k-chemin* $\leftarrow$ KShortestPaths($Graphe_{physique}$, nbrKPath)*//déterminer k plus courts chemins*
3: S(c) $\leftarrow$ nœud source du c
4: T(c) $\leftarrow$ nœud destination du c
5: nbrLambda(d)$\leftarrow$ nombre de longueurs d'onde nécessaires pour router le trafic de d
6: $\lambda_i(d) \leftarrow$ longueurs d'onde utilisées par d dans le routage initial
7: $\lambda_f(d) \leftarrow$ longueurs d'onde utilisées par d dans le routage final
8: **for** e $\in$ graph1 **do**
9: e $\leftarrow$ "INDEX" *// liste vide qui contient les longueurs d'onde déjà utilisées sur e*
10: **for** rq $\in$ ListRequêtes **do**
11: **for** s=0 to |*k-chemin*| by nbrKPath **do**
12: **if** (S(*k-chemin*[s])=S(rq)) and (T(*k-chemin*[s])=T(rq)) **then**
13: listi $\leftarrow$ S(*k-chemin*[s])
14: **for** e $\in$ *k-chemin[s]* **do**
15: listi $\leftarrow$ T(e)
16: ListDemand $\leftarrow$ créer demande(listi)
17: **for** d $\in$ List **do**
18: listLambda $\leftarrow \lambda_i(d)$
19: nbr $\leftarrow$ 0
20: j $\leftarrow$ 0
21: **while** nbr $\neq$ nbrLambda(d) **do**
22: **if** j $\notin$ listLambda **then**
23: listLambda $\leftarrow$ listLambda $\cup$ j
24: $\lambda_i(d) \leftarrow \lambda_i(d) \cup$ j *//ajouter la longueur d'onde utilisée par cette demande*
25: **for** e $\in R_I[d]$ **do**
26: "INDEX" $\leftarrow$ "INDEX" $\cup$ j *//ajouter la longueur d'onde sur les arcs de la route finale de d*
27: nbr $\leftarrow$ nbr+1
28: j $\leftarrow$ j+1
29: **else**
30: j $\leftarrow$ j+1
31: **return** max(listLambda)+1

Le deuxième routage se donne comme but d'améliorer le premier routage en termes de ressources utilisées sans aucune contrainte sur la qualité de service. Ce qui se traduit donc pas la réduction du nombre de longueurs d'onde utilisées pour répondre au demandes. Pour cela, nous utilisons un solveur de programmes linéaires mixtes à savoir CPLEX [1] auquel nous passerons une formulation du problème de multicommodité avec les deux types de contraintes classiques : les contraintes de flot ainsi que les contraintes de capacité induites par le nombre des longueurs d'onde disponibles X.

Pour la recherche du nombre X en un nombre logarithmique d'étapes, nous procéderons comme suit : si le premier routage glouton utilise W longueurs d'onde, nous essayerons de résoudre le programme linéaire avec X=W/2. S'il n y a pas de solution faisable, l'on essayera avec X = W-(W-X)/2 à savoir 3W/4 et ainsi de suite jusqu'à trouver une solution faisable.

Pour rendre la recherche plus rapide nous donnerons comme paramètre une limite de temps pour CPLEX ; ainsi pour chaque X, il considèrera qu'il n y a pas de solution faisable dès que cette limite est dépassée.

Le principe de fonctionnement du routage Glouton est présenté au niveau de l'Algorithme 8. Il utilise la fonction KShortestPaths, celle ci détermine k plus courts chemins pour chaque demande. Une fois la route qu'une demande de connexion doit emprunter pour atteindre sa destination est déterminée, nous déterminons les longueurs d'onde qui sont encore disponibles sur les arêtes de cette route. Ceci nécessite une connaissance de l'état d'occupation du réseau. Nous réservons à chaque demande les longueurs d'onde dont elle a besoin pour le routage de son trafic. S'il n'y a pas suffisamment des longueurs d'onde, on ajoute selon le besoin.

Lors du choix des longueurs d'onde, nous devons respecter la contrainte de continuité en longueur d'onde. C'est le fait de réserver une longueur d'onde sur toute la route, de la source à la destination. Finalement, le nombre des longueurs d'onde nécessaires pour router les demandes W sera le plus grand indice des longueurs d'onde utilisées sur les arêtes du graphe physique plus 1. Ce nombre W sera utilisé ensuite par Cplex pour déterminer le routage final.

Grâce aux deux routages, nous serions capable de générer le graphe de dépendance et lui appliquer l'une de deux approches de traitement, par feuilles ou par niveau.

[1] http ://www-01.ibm.com/software/integration/optimization/cplex-optimizer

4.3.2 Dépendance par Feuilles

Le principe de cette approche est de déterminer les feuilles à chaque itération, sachant qu'une feuille est un sommet du graphe de dépendance qui n'a pas des successeurs. Une sélection est effectuée sur l'ensemble de ces feuilles afin de déterminer quelle requête doit être reroutée à cet instant. Le choix de la requête à rerouter peut se faire selon l'une de deux approches, ou bien celle de la Matrice ou celle du Tri.

Dépendance par Feuilles selon la notion Tri

Algorithm 9 Dépendance par Feuilles selon le Tri : ($Graphe_{physique}$ graph, double α)$\mapsto$ (double c, Tableau ordre[0$\cdots$ m-1])

1: tailleordreOpt $\leftarrow$ 0// variable utilisée pour sauvegarder à chaque itération le niveau dans l'ordre optimal
2: listeVertices $\leftarrow$ contient l'ensemble des feuilles du graphe graph (ensemble des sommets qui n'ont pas des successeurs)
3: ok $\leftarrow$ false *// variable booléenne*
4: niveau $\leftarrow$ 0 *// le niveau d'ajout des requêtes dans l'ordre final*
5: ordre $\leftarrow$ tableau qui contient l'ensemble des feuilles
6: OrdreOpt $\leftarrow$ tableau qui contient l'ordre final de traitement
7: **for** i=0 to $|ordre|$ **do**
8: ordre[i] $\leftarrow$ listeVertices[i]
9: **repeat**
10: **if** $|listeVertices|) \neq 0$ **then**
11: (cout, newOrdre) $\leftarrow$ Ordre via Tri par propagation des demandes(ordre, 0, α)
12: s $\leftarrow$ newordre[0]
13: OrdreOpt[niveau] $\leftarrow$ s
14: niveau $\leftarrow$ niveau+1
15: $V(Graphe_{physique}) \leftarrow V(Graphe_{physique}) \setminus \{s\}$ *//supprimer cette requête du graphe graph*
16: listeVertices $\leftarrow$ les feuilles du graphe graph
17: **else**
18: ok $\leftarrow$ true
19: **until** ok=true
20: cost $\leftarrow$ Coût(OrdreOpt, α)
21: **return** (cost, OrdreOpt)

Dépendance par Feuilles selon la notion SML

Algorithm 10 Dépendance par Feuilles selon SML : ($Graphe_{physique}$ graph, double α)$\mapsto$ (double c, Tableau ordre[0$\cdots$ m-1])

1: tailleordreOpt $\leftarrow$ 0 // *variable utilisée pour sauvegarder à chaque itération le niveau dans l'ordre optimal*
2: fin $\leftarrow$ $|V(Graphe_{physique})|$ // *nombre des sommets dans le graphe graph*
3: **for** s $\in$ V($Graphe_{physique}$) **do**
4: listeVertices $\leftarrow$ contient l'ensemble des feuilles du graphe graph
5: (cout,ordre) $\leftarrow$ Ordre via Matrice 1(graph,α)
6: OrdreOpt[tailleordreOpt] $\leftarrow$ ordre[0]
7: tailleordreOpt $\leftarrow$ tailleordreOpt+1
8: $V(Graphe_{physique}) \leftarrow V(Graphe_{physique}) \setminus$ {s} //*supprimer cette requête du graphe graph*
9: cost $\leftarrow$ Coût(OrdreOpt, α)
10: **return** (cost, OrdreOpt)

4.3.3 Dépendance par Niveau

L'approche de tri par Niveau détermine de même l'ensemble des feuilles du graphe de dépendance à chaque itération. Contrairement à l'approche précédente, la dépendance par Niveau détermine le meilleur ordre pour rerouter l'ensemble des feuilles, elle les reroutent selon cet ordre et passe ensuite au niveau supérieur. Pour ordonnancer les feuilles, on procède soit avec la notion Matrice ou celle du Tri.

Dépendance par Niveau selon la notion Tri

Algorithm 11 Dépendance par Niveau selon le Tri : ($Graphe_{physique}$ graph, double α)$\mapsto$ (double c, Tableau ordre[0$\cdots$ m-1])

```
1: listeVertices ← contient l'ensemble des feuilles du graphe graph
2: t ← 0 // le niveau d'ajout des requêtes dans l'ordre final
3: ordre ← tableau qui contient l'ensemble des feuilles
4: for i=0 to |ordre| do
5:       ordre[i] ← listeVertices[i]
6: (cout, newOrdre) ← Ordre via Tri par propagation des demandes(ordre,
   0, α)
7: vider ordre
8: ordre ← newOrdre
9: level[0] ← listeVertices // ensemble contenant les différents niveaux du
   graphe graph, chaque niveau est un ensemble des sommets
10: i=0 // variable utilisée pour les niveaux dans le graphe
11: repeat
12:       vider listeVertices
13:       for v ∈ level[i] do
14:             listeVertices ← les prédécesseurs de v
15:             if taille(listeVertices) ≠ 0 then
16:                   t ∈ taille(ordre) // c'est à partir de t qu'on va commencer
                      le tri par la suite
17:                   level[i+1] ← listeVertices
18:                   for j=0 to taille(listeVertices) do
19:                         ordre[j+t] ← listeVertices[j]
20:                   (cout, newOrdre) ← Ordre via Tri par propagation des de-
                      mandes(ordre, t, α)
21:                   vider ordre
22:                   ordre ← newOrdre
23:                   listeVertices ← les feuilles du graphe graph
24:                   i ← i+1
25:             else
26:                   ok ← 1
27: until ok=1
28: cost ← Coût(ordre, α)
29: return (cost, ordre)
```

Dépendance par Niveau selon la notion SML

Algorithm 12 Dépendance par Niveau selon SML : ($Graphe_{physique}$ graph, double α)$\mapsto$ (double c, Tableau ordre[0$\cdots$ m-1])

```
1: tailleordreOpt ← 0// variable utilisée pour sauvegarder à chaque itération le niveau dans l'ordre optimal
2: for s ∈ V(graph) do
3:     listeVertices ← contient l'ensemble des feuilles du graphe graph
4:     (cout,ordre) ← Ordre via Matrice 1 : (graph,α)
5:     t ← |listeVertices|
6:     for i=0 to (t-1) do
7:         OrdreOpt[i+tailleordreOpt] ← ordre[i]
8:     tailleordreOpt ← |ordre|
9:     V(graph) ← V(graph) \ {listeVertices } //supprimer toutes les feuilles du graphe graph
10: cost ← Coût(OrdreOpt, α)
11: return (cost, OrdreOpt)
```

4.4 Simulations

Dans cette section, il s'agit de comparer les différentes approches que nous avons proposé précédemment. Nous expliquerons tout d'abord le principe des scénarii réalisés et enfin nous discutons les résultats obtenus.

Nous avons testé les algorithmes de façon plus réaliste. En effet, nous avons considéré la topologie physique du réseau européen Atlanta dont les caractéristiques sont présentées au niveau du tableau 4.1.

Nous avons de même récupéré les demandes et le trafic correspondant à chacune d'entre elles. c'est sont des demandes applicables au graphe physique choisit. Nous avons adapté le trafic supporté par chaque longueur d'onde de manière à ce que chaque requête nécessite au maximum 4 longueurs d'onde pour rerouter son trafic.

TABLE 4.1 – Caractéristiques du réseau européen Atlanta

Caractéristiques du réseau	Valeurs
Nombre de nœuds	15
Nombre des liens	22
Nombre des demandes	210

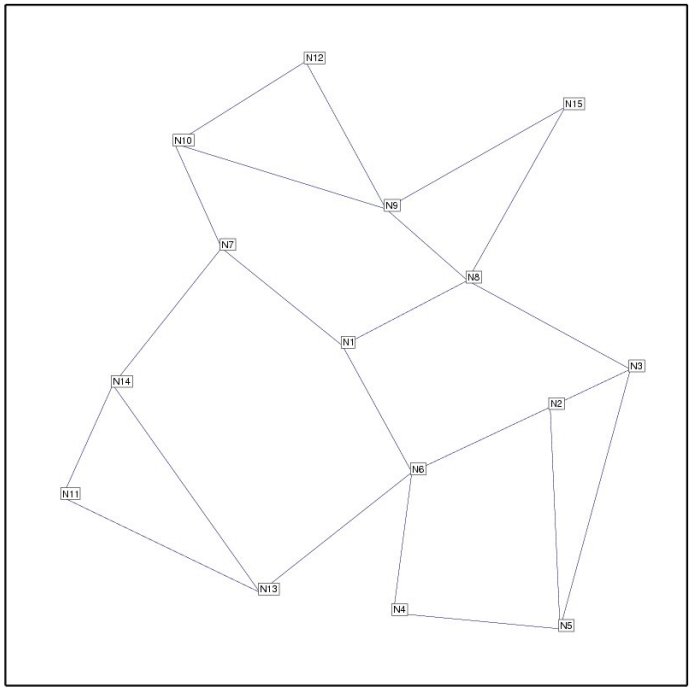

FIGURE 4.3 – Topologie du réseau Atlanta

Dans un premier temps nous avons pris les valeurs de α=[1.5, 3, 5, 7.5, 10], elles sont utilisées pour le traçage de toutes les courbes de simulations. Lors de la reconfiguaration, il a fallut générer les deux routages initial et final R1 et R2. R1 a été généré à l'aide du routage Glouton (voir Algorithme 8). Nous avons obtenu un W=26, c'est le nombre des longueurs d'onde que nécessite ce routage pour pouvoir router le trafic des demandes. Pour R2, Cplex a pu router le même trafic avec 25 longueurs d'onde.

4.4.1 Scénarii de simulations

Dépendance par Feuilles

La dépendance par Feuille présente la première approche du traitement du graphe de dépendance, elle a été détaillée au niveau de la section 4.3.2. Dans le but d'étudier cette approche, nous avons utilisé deux notions développées au niveau de la section 3.4, la SML et le Tri.

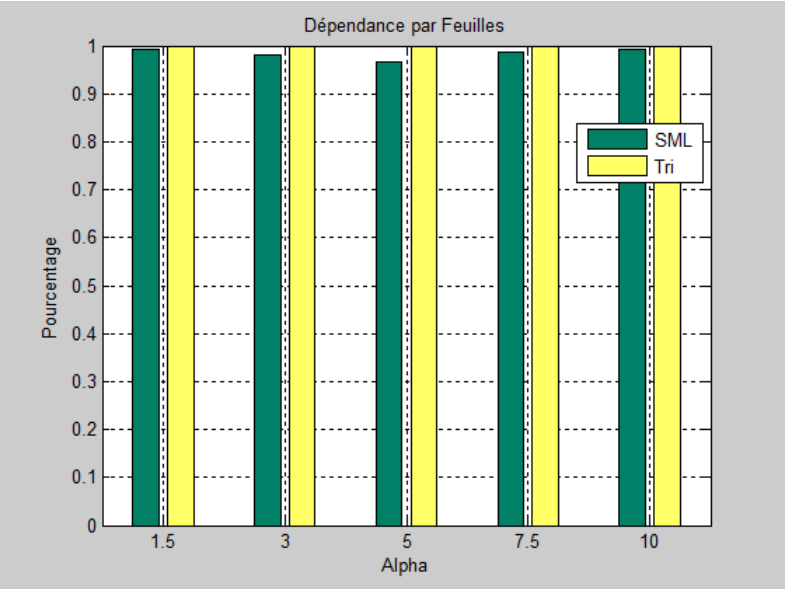

FIGURE 4.4 – Dépendance par Feuilles

La Figure 4.4 représente l'évolution des deux approches pour les différentes valeurs de α. L'histogramme désigné par SML représente le coût récupéré par l'approche qui traite le graphe de dépendance selon la notion de somme minimale des lignes SML. Tandis-que le deuxième histogramme est obtenu suite à l'utilisation de la notion de Tri sur l'ensemble des feuilles.

On constate que la première approche aboutit à un coût meilleur que celui obtenue par la deuxième. Ceci résulte du fait que la notion de SML permet de choisir à chaque itération, laquelle des demandes doit être reroutée. Donc cette approche permet de vérifier la dépendances entre toutes les demandes qui ne sont pas encore reroutées. Par contre, pour l'approche Tri, elle part d'un ordre et elle essaye de le réordonner si c'est possible. Il y a parfois des cas qu'on n'arrive pas à améliorer.

Dépendance par Niveau

Nous procédant de la même manière lors du traitement du graphe de dépendance selon la deuxième approche, la dépendance par Niveau. Nous utilisons les deux notions, la SML et le Tri.

La Figure 4.5 illustre bien que l'approche de SML produit de nouveau un coût meilleur que celui obtenu par l'approche Tri.

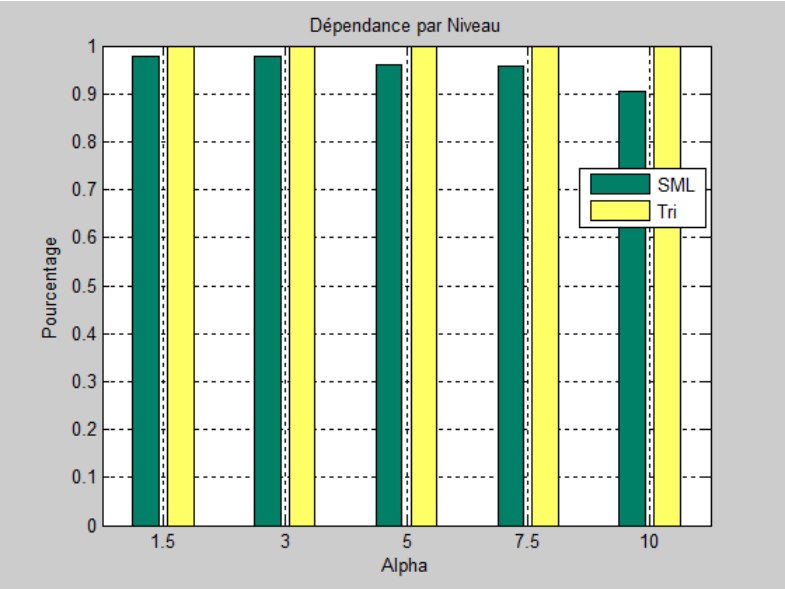

FIGURE 4.5 – Dépendance par niveau

Dans le but de mettre en évidence l'efficacité de ces quatre approches, nous avons tracé les quatre simulations numériques dans une même figure, la Figure 4.6. Les meilleurs résultats ont été obtenus par l'approche de traitement de graphe de dépendance avec la notion Matrice la SML.

Si on fait un retour en arrière sur les simulations obtenues pour le cas des ressources limitées, on trouve que les approches de la matrice, que se soit SML ou SMLNN, produisent toujours un coût meilleur que les autres approches. De même, les résultats obtenus par le traitement du graphe de dépendance que se soit par Feuilles ou par Niveau selon la notion de SML, sont toujours meilleurs que ceux du Tri.

On peut donc tirer comme conclusion que l'approche de traitement selon la notion de la matrice aboutit toujours à un coût meilleur. Ceci est dut en fait, à ce qu'elle teste à chaque itération la dépendance entre toutes les demandes qui ne sont pas encore reroutées. Cette dépendance englobe aussi bien les routes intiales ainsi que les routes finales.

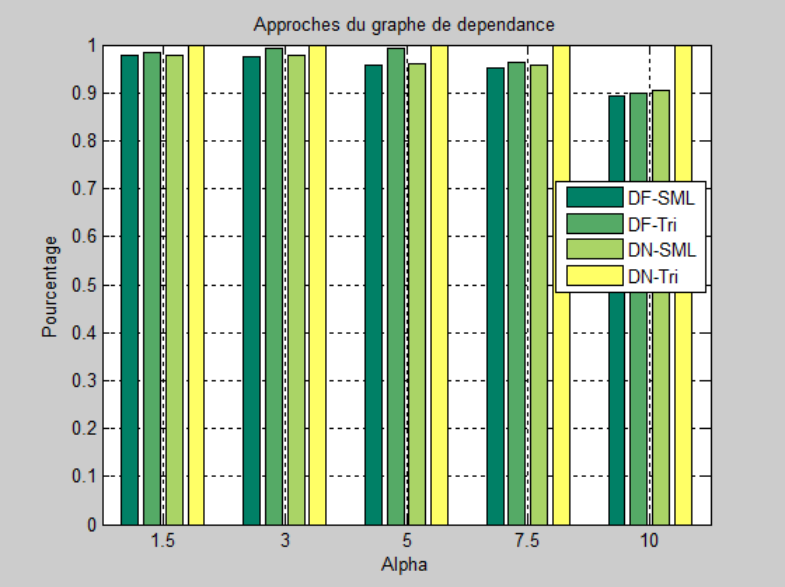

FIGURE 4.6 – Traitement de Graphe de dépendance

Chapitre 5

Conclusions et Perspectives

Dans le cadre de ce projet, nous avons développé des heuristiques et certains algorithmes exactes pour résoudre le problème de reconfiguration du routage dans les réseaux WDM. Ces approches visent à déterminer le meilleur ordre selon lequel seront déplacées les connexions initialement sur le premier routage vers d'autres routes sur second routage.

Nous avons étudié plus particulièrement deux cas lors de reconfiguration du routage. Le premier cas considère qu'il n'y a pas de contrainte sur le nombre de longueurs d'onde, et ainsi les ressources sont toujours disponibles. Dans le second cas, les ressources sont limitées.

Bien que les deux cas soient différents, nous avons pu utiliser des notions développées dans le contexte du premier cas pour mettre en place les algorithmes de traitement de graphe de dépendance du second cas. Des améliorations ont été suggérées pour pouvoir améliorer les performances de nos approches.

Le sujet de reconfiguration du routage est ouvert, il est apte à être traité différemment en fixant une fonction objectif autre que celle adoptée dans ce travail.

Sur le plan de mon projet professionnel, cette étude m'a permis d'enrichir mes connaissances théoriques sur les notions de théorie du graphes, réseaux WDM et reconfiguration du routage. Sur le plan personnel, ce projet m'a permis de renforcer ma créativité, ma détermination et mes capacités d'échange et de travail en groupe. J'espère avoir pu réunir, dans ce rapport, tout ce que j'ai effectué au sein de l'équipe Mascotte à l'INRIA. Enfin, je suis très ravie d'avoir eu l'occasion d'une telle expérience, qui renforce ma motivation à poursuivre dans cette voie.

Bibliographie

[1] D. Coudert. *Algorithmique et optimisation dans les réseaux de télécommunications*. Habilitation à diriger des recherches, Université de Nice Sophia-Antipolis (UNS), March 2010.

[2] D. Coudert, F. Huc, D. Mazauric, N. Nisse, and J-S. Sereni. Reconfiguration of the routing in WDM networks with two classes of services. In *13th Conference on Optical Network Design and Modeling (ONDM)*, Braunschweig, Germany, February 2009.

[3] Elias A. DOUMITH. *Agrégation et Reroutage de Trafic dans les Réseaux WDM Multi-couches*. Doctorat informatique et réseaux, École Nationale Supérieure des Télécommunications, juin 2008.

[4] W. Yao et B. Ramamurthy. Rerouting schemes for dynamic traffic grooming in optical wdm networks. *Computer Networks*, mars 2008.

[5] A. Wason et R.S. Kaler. Rerouting technique with dynamic traffic in wdm optical networks. *Optical Fiber Technology*, octobre 2009.

[6] E. Godard et A. Raspaud G. Fertin. Conversion de longueur d'onde dans les réseaux optiques (facteur d'optimalité 5/3). In *AlgoTel*, 2002.

[7] M. Koubàa. *Routing, Protection and Traffic Engineering in WDM All-Optical Networks*. PhD thesis, École Nationale Supérieure des Télécommunications, 2005.

[8] D. MAQUIN. *Éléments de Théorie des Graphes*. 2003.

[9] R. Paschotta. *Encyclopedia of Laser Physics and Technology, Optical Fiber Communications, Data Transmission, Capacity, Telecom Windows, C Band, L Band, WDM*. 2008.

[10] K. SOL. Le problème du sac à dos. Technical report, Université de Montpellier 2, 2007.