# *Remerciements*

Ceux qui m' ont aidée de près ou de loin durant ces quatre mois à effectuer mon projet dans les meilleures conditions, qu'ils trouvent ici mes sentiments de gratitude et de reconnaissance.

En effet, je tiens à remercier vivement mon encadrant à l'INRIA Mr David COUDERT pour sa disponibilité, sa patience et ses conseils. Je teins aussi à remercier chaleureusement mon encadrant à l'INSAT, Mr Hédi AMARA pour sa disponibilité et ses précieux consignes.

Je remercie aussi toute l'équipe du projet Mascotte, tout particulièrement :

- Mr Jean-Claude Bermond, responsable scientifique du projet.
- Mr Michel Syska.
- Mr Hervé Rivano, attaché de recherche CNRS.
- Mr Fabrice Peix, ingénieur expert.

Je remercie également Mesdames et Messieurs les membres du jury pour l'honneur qu'il m'ont fait de bien vouloir juger ce modeste travail.

Mes remerciements et reconnaissances s'étendent également à l'ensemble du corps enseignant et du personnel de tous les services de l' Institut Nationale des Sciences Appliquées et de Technologie.

# Plan

# Chapitre 1

# INTRODUCTION

Depuis une vingtaine d'années, et encore actuellement, les opérateurs de réseaux d'infrastructures intercontinentaux, trans-continentaux, voire métropolitains privilégient les système de communications optiques utilisant les technologies de multiplexage en longueurs d'onde (**WDM** pour **Wavelength Division Multiplexing**) pour leur bande passante sans équivalent. Pourtant, du point de vue d'un opérateur les investissements nécessaires sont tels que le déploiement de ce type de réseaux ne se fera pas sans garantie d'efficacité suffisante.

En effet, l'optimisation des réseaux optiques **WDM** est essentielle à son développement ; les coûts de déploiement d'un réseau optique d'infrastructure sont gigantesques et les opérateurs sont intéressés par l'optimisation de la taille des liens installés et de la complexité des routeurs présents dans les noeuds du réseau.

Dans ce contexte, l'objectif de ce Projet de fin d'études était de mettre en place une nouvelle approche permettant de mieux utiliser les ressources d'une infrastructure existante, au sein de la bibliothèque d'optimisation **Mascopt**[1]. Ce travail a vocation à permettre aux chercheurs du projet **Mascotte** de pouvoir comparer des algorithmes nouveaux, fruit de leur recherche, à des algorithmes connus.

Pour introduire le contexte technologique de telle problématique le chapitre trois débute par une présentation de la technologie des réseaux optiques tout en définissant ce qu'est le routage optique. Après, et au sein de ce même chapitre, une brève illustration des problèmes classiques du routage ainsi que les solutions qui ont été proposées tout au long des dernières années a été introduite.
!Le chapitre quatre présente une modélisation simple et classique des réseaux optiques tout en définissant les différents problèmes intéressants qu'on peut étudier à travers une telle modélisation. La problématique et le cahier de charge figurent au niveau du chapitre cinq.
L'ensemble du travail, réalisé durant ce PFE, a été présenté au chapitre six .Ce choix a été dicté par le fait qu'on a voulu décrire d'une façon abstraite toutes les phases de notre travail et les différentes solutions proposées suite à des situations bien précises. Ainsi, le chapitre six portera sur tous les modèles mathématiques proposés, les algorithmes, les expérimentations effectuées et les résultats obtenus.

Il est à noter que dans le premier chapitre, une introduction générale du travail à réaliser a été présentée. Le deuxième chapitre esquisse une brève présentation de l'institution et du projet accueillant.

# Chapitre 2

# PRESENTATION DE L'INSTITUTION

## ***2.1- L'INRIA : Institut National de Recherche en Informatique et en Automatique :***

### *2 .1.1- L'organisation :*

Créé en 1967, l'INRIA (alors IRIA) est un établissement public à caractère scientifique et technologique (EPST). Placé sous la tutelle des ministères de la recherche et de l'industrie, il a pour vocation d'entreprendre des recherches fondamentales et appliquées dans les domaines des sciences et des technologies de l'information et de la communication (STIC).

L'INRIA est composé de six unités de recherche situées à Rocquencourt, Rennes, Sophia-Antipolis, Grenoble, Nancy et Bordeaux, Lille, Sarclay et sur d'autres sites à Paris, Marseille, Lyon et Metz. Il y accueille 3500 personnes dont 2700 scientifiques, issus en partie d'organismes partenaires.

En effet, la place que prend l'informatique dans l'ensemble des domaines scientifiques et économiques a conduit l'INRIA à adopter une stratégie de recherche en partenariat avec de nombreux organismes. Ainsi, parmi les 124 équipes (ou projets) de l'institut, plus de la moitié sont des projets communs avec différents organismes de recherches comme le CNRS, les grandes écoles ou les universités. De plus, une grande partie des chercheurs de l'INRIA sont des enseignants et leurs étudiants, qui préparent leur thèse dans le cadre des projets de recherche de l'INRIA.

L'institut travaille également avec de grands laboratoires industriels: les avancées de la recherche en informatique suivent donc de près les besoins en télécommunications, médecine, biologie, transports, etc...

### *2.1.2- Une stratégie de proximité :*

Cette stratégie se traduit par une importante dynamique de transfert des résultats de recherche, aussi bien à travers l'édition, la diffusion de logiciels que par la création de plus d'une soixantaine de start-up soutenue par sa filiale INRIA-Transfert. En effet, cela fait vingt ans que l'INRIA contribue à la création de nombreuses entreprises: de Simulog, créée en 1984, à Genostar technologies, crée en 2004, en passant par Ilog, cotée au Nasdacq ou Kelkoo... Une quarantaine d'entre elles sont encore en activité, d'autres ont intégré de grands groupes industriels.

*2.1.3- Les objectifs :*

Pour l'INRIA, le développement de produits informatiques n'est pas l'objectif premier de la recherche: elle se doit de participer au développement, notamment économique, dans tous les secteurs d'application des STIC. Ainsi, l'évolution des STIC et leurs contributions aux avancées de la science entraînent l'INRIA dans une dynamique de développement au service de l'économie et de la société.

A cette fin, la stratégie de l'institut se décline en sept défis scientifiques prioritaires:

- **Concevoir et Maîtriser** *:* les futures infrastructures des réseaux et des services de communication
- **Développer :** le traitement des informations et des données multimédias
- **Garantir :** la fiabilité et la sécurité des systèmes à logiciel prépondérant
- **Coupler :** modèles et données pour simuler et contrôler les systèmes complexes
- **Combiner :** simulation, visualisation et interaction
- **Modéliser :** le vivant
- **Intégrer :** pleinement les STIC dans les technologies médicales

*2.1.4- Une stratégie internationale :*

Afin de maintenir l'INRIA au plus haut niveau d'excellence, l'institut accorde la plus grande importance à sa stratégie internationale.

Celle-ci se traduit d'abord par les nombreux échanges et partenariats avec des laboratoires étrangers, et par le fait que chaque projet de recherche est évalué par un comité de chercheurs internationaux. De plus, cette stratégie est mise en oeuvre dans le cadre de priorités géographiques: en effet, l'INRIA attache la plus grande importance à son engagement dans le développement d'un espace européen de la recherche.

La deuxième priorité de l'INRIA est l'approfondissement des partenariats avec les laboratoires de recherche aux Etats-Unis car on y trouve la plus grande dynamique de projets de recherche dans le domaine des STIC (environ 40% des projets de l'institut sont en partenariat avec des équipes américaines).

Enfin, une priorité géographique à venir pour l'INRIA est l'Asie, avec laquelle l'institut veut approfondir ses collaborations. Pour cela, l'INRIA s'est engagé dans plusieurs structures conjointes de recherche, par exemple le LIAMA en Chine et le Centre Hong Kong d'ingénierie financière.
L'institut s'engage aussi dans le développement de programmes multilatéraux de recherche avec le Brésil, le Chili, le Mexique et différents pays africains.

## ***2.2- Le projet MASCOTTE:***

**Mascotte** (***Méthodes Algorithmiques, Simulation, Combinatoire et OpTimisation pour les Télécommunications***) est un projet commun entre l'INRIA, le CNRS et l'Université de Nice-Sophia-Antipolis (UNSA), dans le cadre du laboratoire I3S. L'objectif du projet Mascotte est de développer des méthodes et des outils algorithmiques qui s'appliquent en particulier à la conception de réseaux de télécommunication. La réalisation de cet objectif implique la poursuite de recherches de haut niveau dans les domaines de l'algorithmique, de la simulation et des mathématiques discrètes.

### *2.2.1- Axes de recherche :*

- **Algorithmique, mathématiques discrètes et optimisation combinatoire**
- **Algorithmique des communications**
- **Conception et dimensionnement de réseaux (optiques WDM,SDH, ATM, embarqués, radio et satellites)**
- **Simulation orientée objet en environnement réparti**

### *2.2.2- Relations internationales et industrielles :*

Les relations industrielles sont*:*

- projet OSSA (Direction Générale Transport et Energie de la Commission Européenne) sur la simulation de trafic routier.
- Alcatel Espace Industries (Toulouse) sur la conception de réseaux de commutateurs tolérants aux pannes embarqués dans des satellites.
- France Télécom R&D (Contrat de Recherche Collaborative) sur la dimensionnement de services SDH pour les entreprises.
- Projet RNRT (Constellations) sur les constellations de satellites.
- Projet RNRT (Porto) avec Alcatel (Marcoussis) et France Télécom sur la planification et le transport de réseaux optiques.

Les relations internationales sont:

- Brésil: coopération INRIA/CNPq avec l'université de Minas Gerais.
- Italie et Canada: projet avec l'Université de Pise et de Carleton.
- Canada: Simon Fraser University et DalTech.
- Israël: projet avec le Weizmann Institute.
- Espagne: UPC Barcelone.

### *2.2.3- Les logiciels développés :*

Le projet Mascotte a développé plusieurs logiciels, parmi lesquels la librairie d'optimisation de réseaux ***Mascopt*** que j'ai pu utiliser, ainsi que des outils pour la simulation et pour l'analyse

statistique des résultats de simulation (Prosit et Stat-Tool), un logiciel utile pour les réseaux de transport optique (Porto), et un logiciel d'Analyse Technique des Marchés Financiers (InteTech).

*2.2.4- Domaines d'applications :*

Ces cinq dernières années, le projet a choisi comme principal domaine d'application celui des télécommunications, et plus précisément les réseaux de calcul parallèles. Le projet a aussi des applications dans le domaine du transport. On peut noter cependant qu'il y a de fortes similitudes entre ces deux domaines: en particulier, les outils théoriques et les problèmes de communication ne sont pas si différents si l'on considère le domaine du transport ou celui des télécommunications.

Dans le domaine des télécommunications, les applications du projet dépendent beaucoup des partenaires industriels avec lesquels ils sont en collaboration. Avec France Télécom, et les autres partenaires, le projet a travaillé sur le design des réseaux de télécommunications (les réseaux SDH/SONET, WDM, ou encore ATM) et divers problèmes, et sur le regroupement de moyens de transport de faible contenance en d'autres de plus grande contenance. Pour cela, le projet a utilisé les logiciels qu'il a développés, Prosit pour les simulations de trafic routier (dans le cadre du projet européen OSSA), ou encore la simulation d'environnement pour les satellites de télécommunications.

# Chapitre 3

# LES RESEAUX OPTIQUES

La technologie des communications optiques, étudiée dès le début des années 1980 ([1] [2]), est devenue le support privilégié des réseaux de télécommunications. En effet, les caractéristiques des fibres optiques et des émetteurs/récepteurs optoélectroniques actuels s'accordent extrêmement bien aux nécessités des télécommunications modernes.

Les réseaux optiques d'infrastructure se présentent sous la forme de nœuds interconnectés par des liaisons optiques. Les nœuds rassemblent un routeur et des émetteurs-récepteurs. Le routeur fait transiter l'information qui lui arrive d'une fibre ou d'un émetteur vers le récepteur ou la fibre appropriée suivant qu'elle était destinée au nœud lui même ou simplement "en transit".

Les émetteurs/récepteurs sont soit des terminaux (serveurs, ...), soit des réseaux de taille inférieure (par exemple les réseaux d'une ville si l'on considère le réseau d'infrastructure européenne). Les liens optiques sont des câbles comprenant en général plusieurs fibres. En effet, le coût majoritaire de l'installation d'un lien est dû aux travaux d'enfouissement qui sont incompressibles et les opérateurs préfèrent les rentabiliser en installant plusieurs fibres à la fois.

La technologie optique, décrite plus en détail en section 3.1, offre des performances que les émetteurs/récepteurs électroniques actuels sont incapables d'exploiter. Une solution à ce déséquilibre est apportée par la technique de multiplexage en longueur d'onde (WDM), présentée en section 3.2. Les réseaux WDM fonctionnent en mode connecté : à chaque paire émetteur/récepteur voulant communiquer, il faut affecter un chemin optique c'est-à-dire un chemin dans le réseau et une longueur d'onde.

Ce problème, dit du routage optique, est décrit en section 3.3. Il consiste à vérifier si il est possible de réaliser l'ensemble des demandes de communication sur un réseau avec k fibres par lien et l longueurs d'onde disponibles par fibre.

## ***3.1- La technologie optique et le multiplexage WDM :***

Une liaison optique est réalisée principalement à l'aide d'un "Laser", d'une fibre optique conduisant le signal et d'un photo-détecteur. L'émetteur transforme la suite de bits à transmettre en modulation d'une onde optique porteuse. La fréquence de cette onde détermine la bande passante disponible sur le lien car au plus un bit par période peut être transmis.

La bande passante offerte par les communications optiques est leur principal point fort : les fréquences des signaux optiques allant de $10^{14}$ à $10^{15}$ Hz, il est possible d'atteindre, en théorie, la centaine ou le millier de térabits par seconde (1 Tb = $10^{12}$ bits). Par exemple, un

objectif de France Télécom est de diffuser 18000 chaînes de télévision dans le monde en haute définition, ce qui représente des flux d'environ 5 10 Tb / s.

L'autre intérêt des communications optiques est dû aux fibres optiques qui les portent. Celles-ci sont moins volumineuses, plus légères et moins chères que les câbles électriques utilisés dans les réseaux de générations précédentes. De plus elles présentent des taux d'atténuation du signal bien inférieurs à ceux des câbles en cuivre et des taux de pertes microscopiques, même pour de grandes distances : pour des débits allant jusqu'à 1 Gb/s, le signal présente un taux d'erreur considéré comme nul (< 109 erreurs par bit) jusqu'à 200 Km. En plaçant des répéteurs le long d 'un lien, il est facile et peu coûteux d'obtenir des liaisons optiques de plusieurs milliers de kilomètres (comme la fibre Europe Australie).

Ces deux atouts, faibles coûts comparés et faibles taux de pertes, ont permis l'explosion des liaisons intercontinentales.

Un problème se pose alors. La bande passante disponible dans les réseaux modernes est titanesque, mais les consommateurs de cette ressource restent des équipements électroniques (serveurs Web, visioconférence, téléphonie, télévision, ...) qui sont loin de l'exploiter complètement.

Une réponse technologique à ce décalage est le multiplexage des flux d'informations. Les techniques de multiplexage permettent d'exploiter une plus large bande passante de la fibre que celle qu'utilise un seul émetteur. Elles consistent à mélanger plusieurs flux d'information qui seront transmis sur la même fibre. La juxtaposition des flux peut se faire temporellement sur une même onde (multiplexage temporel) : l'onde est découpée en périodes de courte durée, sur lesquelles chaque flux est codé. La modulation de l'onde pendant cette durée se fait selon le flux d'information correspondant. Ce type de multiplexage trouve ses limites dans le coût prohibitif des équipements et dans la lenteur des systèmes optoélectroniques existants.

La technologie du multiplexage en longueur d'onde ***WDM*** (pour ***Wavelength Division Multiplexing***) résout ce problème au prix d'une plus grande complexité d'optimisation. Cette technique est le standard actuel dans les réseaux optiques. Elle consiste à transférer simultanément plusieurs signaux de longueurs d'onde différentes à travers de nombreuses fibres optiques.

La modulation d'une onde se fait de la manière classique. Les ondes sont ensuite multiplexées et démultiplexées par des systèmes optiques passifs s'apparentant au prisme.
L'avantage de ce type de multiplexage est qu'il n'est pas tributaire de la vitesse des équipements électroniques. Cependant le nombre de longueurs d'onde qu'une fibre peut transporter est borné, même s'il a augmenté de façon surprenante ces dernières années : d'une dizaine de longueurs d'ondes à moyen débit il y a quelques années, on est passé à plusieurs milliers de longueurs d'onde à très haut débit.

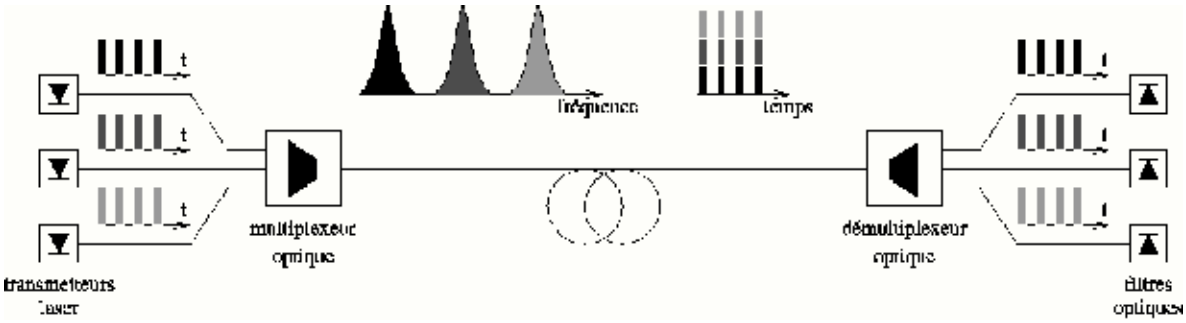



**Figure3.1 : La technique de multiplexage en longueurs d'onde**

## *3.2- Fonctionnement des réseaux optiques WDM :*

Les réseaux optiques fonctionnent pour la plupart en mode connecté : lorsqu'un émetteur veut se mettre en communication avec un récepteur, un chemin optique doit être assigné à la communication. Un chemin optique est décrit par la donnée d'une suite de fibres allant de l'émetteur au récepteur et de la longueur d'onde utilisée sur chaque fibre. Cet ensemble de données constitue le routage de la communication.

L'affectation des longueurs d'onde doit satisfaire une contrainte forte : deux chemins optiques utilisant la même fibre ne peuvent être porté par la même longueur d'onde. Dans le cas contraire, les signaux se brouilleraient mutuellement et les informations qu'ils transportent seraient perdus.

Certains routeurs permettent de faire de la conversion de longueur d'onde, c'est-à-dire permettent à un chemin qui rentre dans le nœud avec une certaine longueur d'onde d'en ressortir avec une autre. On peut distinguer deux types de schéma de conversion :

- ***la conversion totale*** où un chemin utilisant une longueur d'onde quelconque en entrée peut ressortir avec n'importe quelle autre longueur d'onde (toujours sous contrainte qu'il n'y ait pas deux chemins utilisant la même longueur d'onde sur la même fibre);

- et ***la conversion partielle*** où une longueur d'onde particulière ne peut être convertie qu'en une longueur d'onde appartenant à un ensemble dépendant de la longueur d'onde de départ.

Ces deux types de routeurs existent, mais dans la suite de notre travail, nous ne considérerons que des réseaux dits « transparents », c'est-à-dire sans conversion en longueur d'onde.

En théorie, une fibre peut-être parcourue par un signal dans les deux sens. En pratique toutefois, des contraintes technologiques dues au placement des équipements d'émission et de réception des nœuds, imposent des communications unidirectionnelles. Par ailleurs, le coût d'une fibre supplémentaire entre deux nœuds étant minime, les réseaux sont la plupart du temps symétriques. Nous considérerons donc par la suite des réseaux multifibres symétriques. Nous considérons aussi que tous les liens ont le même nombre de fibres. Cette simplification reste

proche de la réalité puisque les fibres sont regroupées dans des câbles de tailles standardisées. Un tel réseau est dit « ***k-fibres*** », où k est le nombre de fibres par lien.

### ***3.3- Routage optique :***

Dans le cas des réseaux optiques WDM, on appelle demande de communication un couple (émetteur, récepteur) qui doit être connecté par un chemin dans le réseau. Un ensemble de demandes de communication est appelé une instance de communication.

Réaliser une instance de communication consiste à trouver un routage optique, i.e. un multi ensemble de chemins reliant chaque demande de communication de l'instance et une affectation de longueurs d'onde à ces chemins satisfaisant à la contrainte des réseaux WDM : deux chemins traversant une même fibre, ne peuvent pas se voir affecter la même longueur d'onde et un chemin doit utiliser une seule longueur d'onde tout au long des fibres parcourues (***contrainte de continuité de longueur d'onde)***.

Dans ce contexte, le problème que les opérateurs voudraient voir résoudre est le suivant : étant données la topologie d'un réseau (c'est-à-dire l'ensemble des nœuds et des câbles reliant ces nœuds) et une instance de communication, trouver le réseau le moins cher tel qu'il soit possible d'y réaliser l'instance. Mais avec la contrainte de ***« continuité de longueur d'onde »*** le problème se distingue sûrement plus compliqué.

Dans la littérature, le routage optique est connu aussi sous le nom de « ***Problème d'affectation de chemin et de longueur d'onde ».*** Et cela vient du fait que pour un ensemble de demandes de connexions qui ont besoin d'être établies et un ensemble de contraintes relatives au nombre de longueur d'onde, il est nécessaire de déterminer les routes sur lesquels les connexions vont être établies et les longueurs d'onde qui vont être affectées à ces connexions.

Pour être plus précis considérons un exemple illustratif d'attribution de longueur d'ondes pour un ensemble de connexions bien définie :

Soit l'ensemble de demandes de connexions suivantes :

« $A \rightarrow B$ », « $A \rightarrow D$ » et « $C \rightarrow D$ » définies sur le réseau suivant :

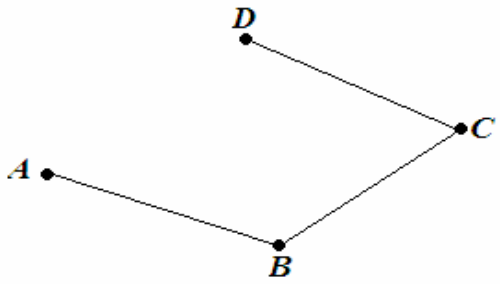



**Figure 3.2: exemple de réseau**

Sur chaque lien du réseau, 2 longueurs d'onde sont disponibles ***λ1*** et ***λ2.***

Avec peu d'attention ou avec une séquence d'arrivée de demandes définie comme suit : « $A \rightarrow B$ » après « $C \rightarrow D$ » et enfin « $A \rightarrow D$ » une attribution possible sera (figure : 2):

- ***λ1*** pour la demande de connexion « $A \rightarrow B$ ».
- ***λ2*** pour la demande de connexion « $C \rightarrow D$ ».
-

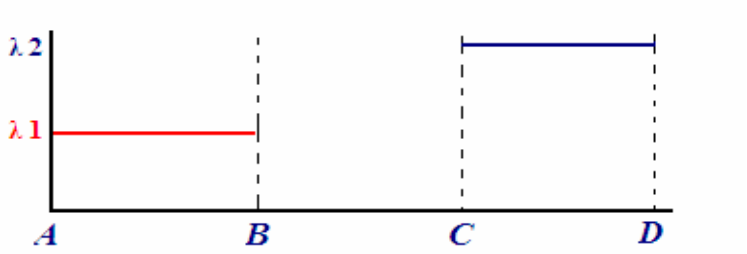



**Figure 3.3: 1er attribution de longueurs d'onde pour les connexions**

Et on ne pourra plus assurer la mise en place de la demande de connexion « $A \rightarrow D$ » suite à la non disponibilité d'aucune longueur d'onde sur les liens composants le chemin entre ***A*** et ***D***.

Une attribution assurant le routage de toutes les demandes de connexion sera par exemple la suivante :

- ***λ1*** pour la demande de connexion « $A \rightarrow B$ ».
- ***λ1*** pour la demande de connexion « $C \rightarrow D$ »
- ***λ2*** pour la demande de connexion « $A \rightarrow D$ ».

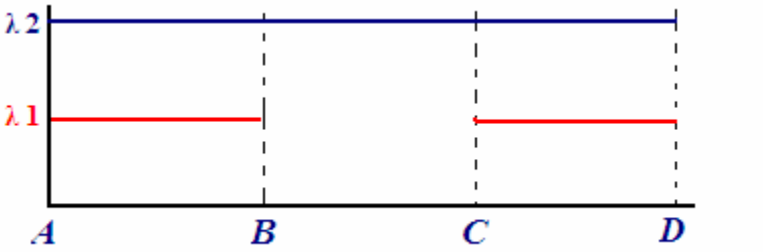



**Figure 3.4: 2eme attribution de longueurs d'onde pour les connexions**

*3.3.1 – Les classes de trafic :*

Typiquement, [3] on distingue trois classes de trafic :

1. **Trafic statique** *:* Toutes les connexions sont connues en avance donc le problème sera ramené à l'établissement d'une *route-optique* avec un minimum de ressources réseau utilisées (les L.O et les fibres).
   Ce problème est connu sous le nom de *SLE : Static Lightpath Establishment.*

2. **Trafic incrémental** : Les requêtes arrivent séquentiellement et une *route-optique* est attribuée pour chaque connexion et reste indéfiniment sur le réseau.

3. **Trafic dynamique** : Une *route-optique* est établie juste à l'arrivée d'une connexion et elle sera libérée après une période de temps.

Au niveau de ces deux derniers types de trafic, l'objectif est d'assurer le maximum de connexions établies sur le réseau à un moment donné ou réduire au minimum le taux de blocage et de rejets des demandes de connexion. Ces problèmes sont connus sous le nom de *DLE : Dynamic Lightpath Establishment.*

Pour les problèmes SLE, qui sont des problèmes *NP-complet*, des algorithmes sont utilisés tandis que pour les problèmes DLE qui sont plus difficiles, généralement ce sont des méthodes heuristiques qui sont employées. Et la problématique, pour les deux, pourra être divisée en deux sous problèmes (qui vont être présentés dans la section suivante) :

1- Problème du routage
2- Problème d'affectation de longueur d'onde.

*3.3.2– Problème du routage :*

On peut distinguer trois approches basique de routage dans la littérature [4], [5], [6], [7]:

- **Routage fixe** : La plus simple des approches pour router une connexion est de toujours prendre le même chemin, en fait les routes sont calculées pour les paires source/destination en OFF-LINE et à l'arrivée de chaque requête pour une paire <s,d> un chemin <p> pré calculé va être assigné.

Avantage : Simplicité.
Inconvénients : Grande probabilité de blocage suite à une congestion ou panne sur un lien.

- **Routage fixe/auxiliaire** : Chaque noeud doit maintenir une table de routage contenant un ensemble de chemin pour chaque paire ***<s,d>***, un primaire et les autres auxiliaires.

Avantage : Dépassement du problème des défauts sur les liens et réduction de la probabilité de blocage relativement au routage précèdent pour des réseaux spécifiques.

- **Routage adaptative** *:* Dans ce cas la route est choisie dynamiquement suivant l'état du réseau.

Avantage : une probabilité de blocage moindre que les deux exemples précédents.
Inconvénients : une complexité de calcul qui se greffe.

### *3.3.3 - Problème d'affectation de L.O :*

Le problème d'affectation de longueurs d'ondes est décomposé en deux grands problèmes [3], [9], [10]:

#### *3.3.3.1- Le problème d'affectation statique de longueurs d'onde :*

Comme il a été dit précédemment, les demandes de connexions sont connues à l'avance et les opérations de routage et d'attribution de longueur d'ondes sont exécutées en mode OFF-LINE (avant l'arrivée des demandes). Généralement, l'objectif d'un tel problème est de minimiser le nombre de ressources (L.O) nécessaires pour la mise en place d'un ensemble de connexions sur une topologie physique bien définie. Un autre objectif peut être la maximisation de nombres de demandes de connexions acceptées, ainsi les solutions auront tendances à établir des connexions plus courtes traversant un nombre réduit de liens optiques.

Etant donné un ensemble de demandes de connexions, une topologie de réseau physique et un ensemble de L.O disponibles. Un chemin est choisi pour chaque demande de connexion, ainsi le nombre de connexions traversants un lien physique défini la congestion sur ce lien. Il est à noter que les L.O doivent être attribuées de telle façon que 2 connexions qui partagent le même lien physique ne partagent jamais la même LO.

#### *3.3.3.2- Le problème d'affectation dynamique de longueur d'onde :*

Dans cette partie dix heuristiques vont être présentées :

**-1- Random Wavelength Assignement :**

Cette heuristique s'exécute en deux étapes, en premier lieu un ensemble de longueurs d'ondes disponibles sur la route spécifiée est cherchée, et en second lieu un tirage aléatoirement de cet ensemble va servir la connexion sur cette route.

**-2- First Fit :**

Dans ce modèle, toutes les longueurs d'ondes sont numérotées et la recherche d'une L.O disponible sur une route spécifiée se fait en considérant celles d'indices minimum en premier lieu (recherche dans un ordre croissant : minimisation de l'espace de recherche). La première LO disponible sera prise, l'idée derrière une telle approche

est d'utiliser tant que possible une portion de l'espace des longueurs d'ondes pour servir les demandes de connexions et ceci dans le but d'assurer la disponibilité de L.O pour des connexions ayant des chemins assez long et qui va assurer dans une autre vision la minimisation de probabilité de blocage.

**-3- Least-Used/SPREAD :**

Cette approche choisit la L.O qui est la moins utilisée sur le réseau, afin d'équilibrer la charge sur l'ensemble des L.O. Cette approche finira par épuiser rapidement toutes les L.O disponibles sur les longs chemin, ainsi le réseau ne pourra plus que servir les demandes de connexions utilisant un ensemble réduit de liens.

**-4- Most-Used/PACK :**

Cette approche est l'opposé de celle présentée précédemment, en effet la L.O choisie est celle la plus congestionnée.

**-5- Min-Product :**

Cette approche est similaire à celle du First Fit mais la seule différence réside dans la nature du réseau d'application : MP est utilisée dans les réseaux multifibres tandis que FF dans un réseau à fibre unique.

**-6- Least Loaded :**

Désigné aussi pour les réseaux multifibres, cette heuristique choisit la L.O qui a la plus grande valeur de capacité résiduelle sur le lien le plus chargé d'une route spécifiée.

**-7- MAX-SUM :**

Proposé pour les réseaux multifibres mais peut être appliqué aux réseaux mono-fibre. MS considère tous les chemins possibles (les connexions ave leurs chemins présélectionnées) sur le réseau et essaye de maximiser la capacité restante sur chemin après l'établissement d'une connexion. Cet heuristique assume que la matrice du trafic est connue à l'avance, et que la route pour chaque connexion est présélectionnée.

**-8- Relative Capacity Loss** : (Meilleur performance/Implémentation coûteuse).

Basé sur MS (MS peut être vu comme une approche permettant la minimisation de capacité perdu sur les chemins optiques), RCL est lancé suite à l'observation que la minimisation de la capacité totale perdu ne mène pas au meilleur choix, ainsi cette approche calcule la capacité relative pour chaque chemin sur chaque L.O disponible et choisi alors la L.O qui minimise la somme des pertes de capacités sur tous les chemins.

Toutes les heuristiques présentées précédemment ont généralement pour objectif de minimiser le taux de rejet de connexions. Cependant, en considérant que les connexions nécessitant de long chemins ont une plus grande probabilité de rejet alors d'autres schémas ont été proposé pour essayer de protéger les chemins assez long. Ces approches sont :

**-9- Wavelength Reservation.**
**-10- Protecting Threshold.**

Contrairement aux autres approches, ces 2 schémas :

- Ne spécifient pas la L.O à choisir, mais spécifient si une demande de connexion peut avoir une L.O ou non en se basant sur l'instance courante de l'utilisation des L.O sur le réseau.
- Ne peuvent pas fonctionner seules et ils doivent être combinés avec d'autres approches d'attribution de L.O.
- Leur objectif primordial est de protéger seulement les connexions traversant un nombre assez grand de liens (des chemins longs).

Durant plusieurs années, les travaux de recherches ont porté sur le problème RWA, comme présenté précédemment en deux sous problèmes majeurs: le routage et l'affection des longueurs d'ondes. Or chacun de ces problèmes est difficile à résoudre. De ce fait, de nombreuses heuristiques ont été proposées pour résoudre d'abord le problème du routage et ensuite le problème de l'affectation de longueurs d'ondes.

Cette approche est spécifique aux réseaux WDM ainsi les solutions présentés sont dédiées aux réseaux WDM. Cette restriction du modèle a amené la communauté scientifique à choisir une représentation plus générale, c'est-à-dire qui peut s'appliquer à une variété de réseaux (et en particulier les réseaux WDM). Ce modèle appelé, ***« modèle du graphe à niveaux »,*** va être présenté dans le chapitre suivant.

# Chapitre 4

# MODELISATION DES RESEAUX OPTIQUES ET PRESENTATION DES PROBLEMES ETUDIES POUR UNE TELLE MODELISATION

Avant de présenter le modèle, on va introduire quelques notions, utiles pour la suite, sur la théorie des graphes.

## ***4.1- Les graphes :***

Un graphe [8] est une **représentation symbolique** d'un réseau. Il s'agit d'une abstraction de la réalité de sorte à permettre sa modélisation. Principalement un *graphe* permet de décrire un ensemble d'objets et leurs relations, c'est à dire les liens entre les objets.

- Les objets sont appelés les *noeuds*, ou encore les *sommets* du graphe.
- Un lien entre deux objets est appelé une *arête* ou un *arc*.

Un **graphe** ***G*** est un couple ***(V,E)*** où

- ***V*** est un ensemble (fini) d'objets. Les éléments de ***V*** sont appelés les **sommets** du graphe.
- ***E*** est sous-ensemble de ***V*x*V***. Les éléments de ***E*** sont appelés les **arêtes** du graphe.

Une arête ***e*** du graphe relie deux sommets x et y. Les sommets ***x*** et ***y*** sont les **extrémités** de l'arête.

Exemple :

Un exemple de graphe à 8 sommets, nommés ***a*** à ***h***, comportant 10 arêtes :

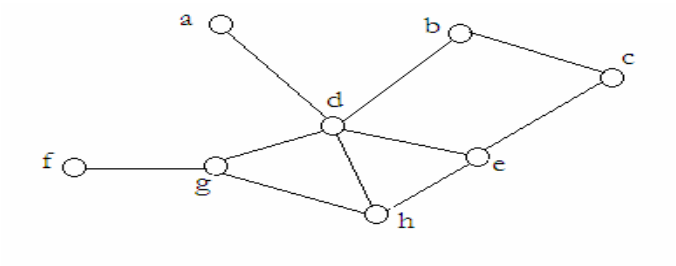



**Figure 4.1: Exemple de graphe avec 8 sommets**

- *G=(V,E)*
- *V={ a, b, c, d, e, f, g, h }*
- *E={ (a,d), (b,c), (b,d), (d,e), (e,c), (e,h), (h,d), (f,g), (d,g), (g,h) }*

Cette définition de graphe correspond au cas des graphes *simples*, pour lesquels il existe au plus une arête liant deux sommets. Dans le cas contraire le graphe est dit multiple.

### ***4.2- Modélisation des réseaux optiques :***

Un réseau WDM est généralement représenté [10] comme suit :

- $G(V,E,F_S,W)$ : où V est l'ensemble des nœuds du réseau, E est l'ensemble des liens bidirectionnelles et W est l'ensemble des longueur d'ondes disponibles par fibre. Supposons que l'ensemble de L.O est le même sur chaque fibre, exemple : $\{\lambda_1, \lambda_2 \ldots, \lambda_{/w/}\}$. Pour chaque lien: $ij \in E$, $F_{ij}$ représente les fibres existantes sur ce lien. Ainsi $ij$ est composé par $|F_{ij}|$ fibres unidirectionnelles et $F_S = \{ F_{ij} \}$.

Exemple de réseau WDM :

Soit le réseau suivant : $G(V,E,F_S,W)$ :

- $V$: {A,B,C,D}
- $E$: $\{A \to B, A \to C, B \to C, D \to B, C \to D\}$
- $W$: 3.
- $F_S$: { $F_{AB}$, $F_{BA}$, $F_{BC}$, $F_{CB}$, $F_{AC}$, $F_{CA}$, $F_{BD}$, $F_{DB}$, $F_{CD}$, $F_{DA}$ } on prend le cas où on a deux fibres dans chaque sens sur un lien donné.

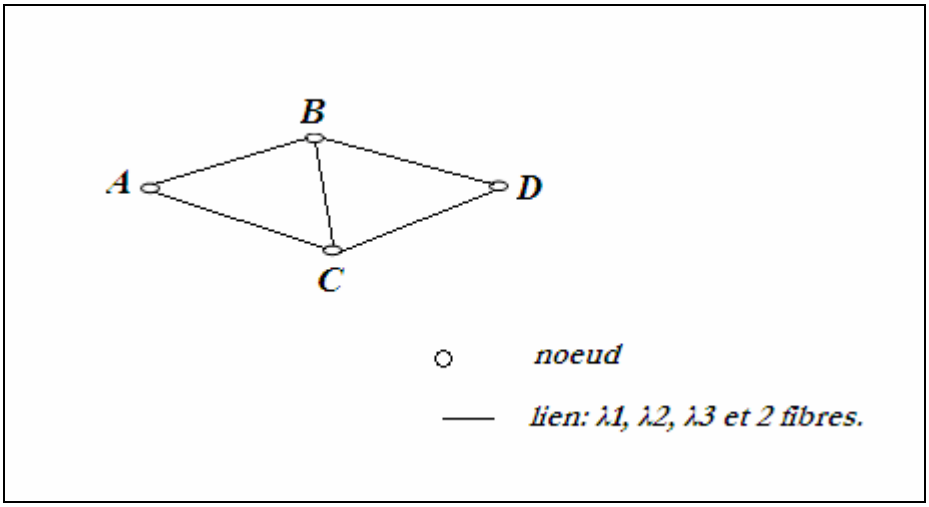



**Figure 4.2: exemple topologie d'un réseau optique**

Appliqué à un tel réseau, le ***modèle du graphe à niveaux*** peut être vu comme suit :

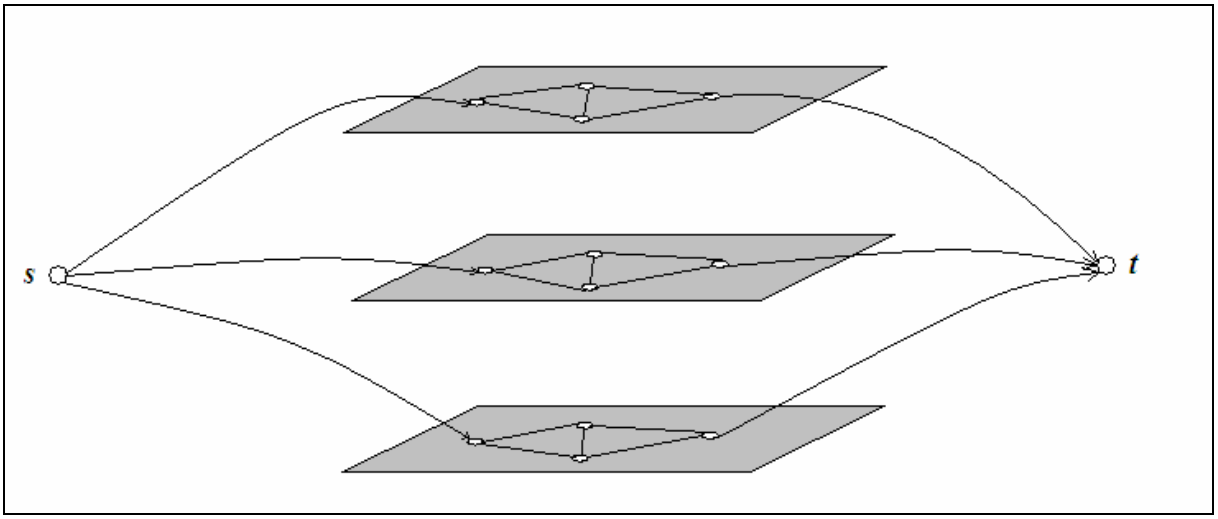



**Figure 4.3 : Représentation en modèle en couches d'un réseau optique**

Ce modèle est décomposé en |W| couches, chacune correspond à une longueur d'onde, et au niveau de chaque couche une réplication de la topologie physique est créée, ou en d'autres termes un graphe avec une représentation plus simple est obtenu : en effet, ce graphe peut être représenté simplement par ***G(V,E,c)*** :

Avec - V : l'ensemble des nœuds.
- E : l'ensemble des liens bidirectionnels sur le graphe.
- c : $\{c_{ij}, ij \in E\}$ est la capacité disponible sur le lien « ***ij*** » (correspond au nombre de fibres disponibles sur ce lien).

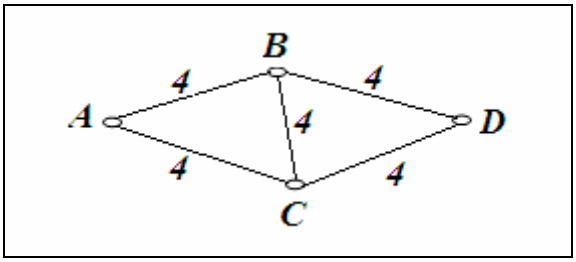



**Figure 4.5 : Représentation d'une couche**

Ainsi, router une demande de connexion entre deux nœuds du graphe de la figure n°1 revient simplement à trouver un chemin, suivant une certaine politique de routage, sur l'un des niveaux du graphe tant qu'il y'a bien sûr assez de capacité.

Durant tout le travail qui va suivre, on ne va plus s'intéresser au problème d'attribution de longueur d'onde, tout notre effort portera essentiellement à assurer l'acceptation de maximum de demandes de connexions sur une couche donnée.

## ***4.3- Chemin dans un graphe:***

Dans un graphe il est naturel de vouloir se déplacer de sommet en sommet en suivant les arêtes. Une telle marche est appelée une *chaine* ou un *chemin*. Un certain nombre de questions peuvent alors se poser : pour deux sommets du graphe, existe-t-il un chemin pour aller de l'un à l'autre? Quel est l'ensemble des sommets que l'on peut atteindre depuis un sommet donné? Comment trouver le plus court chemin pour aller d'un sommet à un autre?

### *4.3.1- Définition d'un chemin sur un graphe:*

- Un **chemin** est une liste p = ($x_1$, … , $x_k$) de sommets telle qu'il existe dans le graphe une arête entre chaque paire de sommets successifs. $\forall i = 1,...,k-1 \quad (x_i, x_{i+1}) \in E$ (ce qui revient évidement à une succession de liens).
- La **longueur** du chemin correspond au nombre d'arêtes parcourues : ***k-1***.

Exemple :

Un chemin de longueur ***5*** dans le graphe reliant les sommets ***f*** à ***b***.

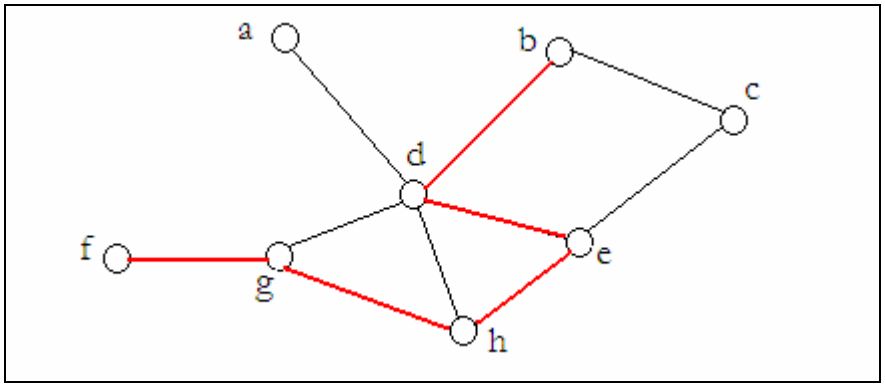



**Figure 4.5 : Chemin sur un graphe**

***- P = ( f, g, h, e, d, b )***

Il existe bien d'autres chemins pour aller de ***f*** à ***b*** : par exemple ***(f, g, d, b)*** de longueur 3, le *chemin* ***(f, g, d, h, e, d, b)*** de longueur 6, ou encore ***(f, g, d, h, e, d, h, e, d, b)*** de longueur 9, ... ***(d, h, e, d)*** est appelé un cycle. Ce cycle pouvant être emprunté autant de fois que l'on veut, il y a un nombre infini de chemins de ***f*** à ***b.***

Lorsqu'un chemin existe entre deux sommets dans un graphe, l'être humain se pose rapidement la question non seulement de trouver un tel chemin, mais bien souvent il est intéressé par le *plus court chemin* possible entre ces deux sommets.

On dispose de plusieurs méthodes systématiques qui peuvent, pour tout graphe et pour toute paire de sommets ***s*** et ***t***, déterminer le plus court chemin entre eux. Résoudre ce problème va donc consister à proposer des algorithmes, aussi rapides que possible.

Comme il a été indiqué précédemment, toute la richesse du graphe réside au niveau de ses arêtes, en d'autres termes les significations que puissent avoir ses arêtes. Ainsi, la notion de plus court chemin sera généralisée dans le cas de graphes valués, où chaque arête ***e*** est associée à une valeur ***c(e)***, appelée souvent son poids. La valuation des arêtes peut représenter des coûts de transit, des distances kilométriques, le temps nécessaire pour parcourir les arêtes,... .

### *4.3.2- Définition du plus court chemin et présentation de l'algorithme de Dijkstra :*

Dans un graphe valué, on cherche à déterminer les plus courts chemins d'un sommet racine (source) ***r*** aux autres sommets. Ici ***c(p)*** représente la somme des poids des arêtes le long d'un chemin ***p***. le résultat peut se donner sous forme d'une fonction poids et d'un arbre ***T*** appelé arbre des plus courts chemins tel que $\forall x \in V(G), dist_G(r,x) = dist_T(r,x)$.

Différents algorithmes pour trouver les plus courts chemins sur un graphe valué ont été proposés mais les plus importants sont :

- ***Bellman Ford***.
- ***Dijkstra***.

- *Algorithme de **Dijkstra** :*

L'algorithme le plus connu est du à **Dijkstra** ; il est basé sur le principe suivant : soit $S \subset V(G)$ contenant *r* et soit $\overline{S} = V(G) \setminus S$. Si $P = (r, s_1, ..., s_k, \overline{s})$ est le plus court chemin de r à $\overline{s}$ alors $s_k \in S$ sinon le chemin de *r* à $s_k$ serait un chemin plus court. De plus le sous chemin de $P = (r, s_1, ..., s_k)$ est un plus court chemin de *r* à $s_k$. On a donc :

$$dist_G(r,\overline{s}) = dist_T(r,s_k) + p(s_k\,\overline{s})$$

Et la distance de r à $\overline{s}$ est donné par la formule :

$$dist_G(r,\overline{s}) = \min_{u \in S, v \in \overline{S}} \{dist(r,u) + p(uv)\}$$

L'algorithme construit en partant de $S_0$ = r une suite d'ensembles $S_1$, $S_2$, $S_3$, …, $S_i$ de sorte qu'à l'étape i on a trouvé les plus courts chemins de r aux sommets de $S_i$. on appellera $T_i$ l'arbre partiel de ces plus court chemins.

Afin d'éviter trop de calculs, tout au long de l'algorithme, on associe à chaque sommet $v \in V(G)$ une fonction $d'(v)$ qui est borne supérieure pour $dist(r,v)$ et un sommet $p(v)$ qui est le père potentiel de *v* dans l'arbre.

A chaque étape *i*, nous avons :

$$d'(v) = dist(r,v) \qquad \text{si} \quad v \in S_i$$
$$d'(v) = \min_{u \in V(T_{i-1})} \{dist(r,u) + p(uv)\} \qquad \text{si} \quad v \in \overline{S_i}$$

Ainsi l'algorithme est présenté comme suit :

1. Initialiser $d'(v) = 0$ et $d'(v) = +\infty$ si $v \neq r$ . $T_0 = S_0 = r$ , $u_0 = r$ et $i = 0.$
2. Pour tout $v \in \overline{S_i}$ , si $d'(u_i) + p(u_i v) \leq d'(v)$, faire $d'(v) = d'(u_i) + p(u_i v)$ et $p(v) = u_i$ .Calculer $\min_{v \in \overline{S_i}} d'(v)$ . Soit $u_{i+1}$ un sommet pour lequel le minimum est atteint. Soit $S_{i+1} \cup u_{i+1}$ et $T_{i+1}$ l'arbre obtenu en ajoutant l'arête $p(u_{i+1})u_{i+1}$ à $T_i$ .
3. Si $i = |V| - 1$, rendre T, sinon i = i + 1 et aller à l'étape 2.

Il existe une autre manière pour trouver le plus court chemin entre une source et une destination, mais pour la présenter il faut introduire quelques notions sur le sujet d'***optimisation combinatoire.***

## ***4.4- Optimisation combinatoire :***

Un problème d'optimisation combinatoire [11] consiste à trouver la ***meilleure*** solution dans un ***ensemble discret*** dit ensemble des solutions réalisables. En général, cet ensemble est fini mais de ***cardinalité très grande*** et il est décrit de manière implicite, c'est-à-dire par une liste, relativement courte, de contraintes que doivent satisfaire ***les solutions réalisables***.

Pour définir la notion de *meilleure solution*, une fonction, dite ***fonction objectif***, est introduite. Pour chaque solution, elle renvoie un réel et la meilleure solution (ou *solution optimale*) est celle qui minimise ou maximise la fonction objectif. Clairement, un problème d'optimisation combinatoire peut avoir plusieurs solutions optimales.

**Exemple :**

Le problème du plus court chemin entre deux sommets *A* et *B* d'un graphe est un exemple classique de problème d'optimisation combinatoire. L'ensemble des solutions réalisables est l'ensemble des chemins entre *A* et *B* tandis que la fonction objectif est la longueur du chemin.

*4.4.1- Résolution des problèmes d'optimisation combinatoire :*

Trouver un solution optimale dans un ensemble discret et fini est une trivialité d'un point de vue mathématique. D'un point de vue informatique et pratique, le temps de recherche de la solution optimale est un facteur très important et c'est à cause de lui que les problèmes d'optimisation combinatoire sont réputés si difficiles. La résolution des problèmes d'optimisation combinatoire se heurte donc à ***la théorie de la complexité***. Ainsi, comme l'ensemble des solutions réalisables est défini de manière implicite, il est parfois très difficile de trouver ne serait-ce qu'une solution réalisable.

Quelques problèmes d'optimisation combinatoire peuvent être résolus (de manière exacte) en temps polynomial par exemple par :

- Un ***algorithme glouton***.
- Un algorithme de ***programmation dynamique***.
- Le fait de trouver que le problème peut être formulé comme un ***programme linéaire*** en variables réelles.

Dans la plupart des cas, le problème est ***NP complet*** et, pour le résoudre, il faut faire appel à des algorithmes de « ***branch and bound »***, à la « ***programmation linéaire en nombres entiers »*** ou encore à « ***la programmation par contraintes »***. En pratique, on se contente très souvent d'avoir une solution approchée, obtenue par une « ***heuristique »*** ou une « ***métaheuristique »***. Pour certains problèmes, on peut prouver ***une garantie de performance***, c'est-à-dire que l'écart entre la solution obtenue et la solution optimale est borné.

Pour la suite des problèmes qu'on va traiter tout au long de notre travail, on va se contenter d'utiliser la programmation linéaire comme outil de résolution.

*4.4.2- Quelques notions sur la théorie de complexité:*

Certains problèmes sont plus difficiles que d'autres. Par exemple, étant donné un nombre, il est très facile de tester si ce nombre est divisible par 2 ou non : il suffit de regarder si le dernier chiffre appartient à l'ensemble {0,2,4,6,8}. En revanche, il est beaucoup plus difficile de déterminer sa décomposition en produits de facteurs premiers.

**Définition :** Un algorithme est dit en temps *polynomial* si, pour tout n, pour des données ne prenant pas plus de n octets, l'algorithme s'exécute en moins de « $C.n^k$» opérations élémentaires (les constantes C et k étant bien sûr indépendantes de n).

Dans la définition précédente, le terme d'opérations élémentaires est assez vague. On doit le comprendre comme des additions, des multiplications, des comparaisons, etc... (tout ce que le processeur sait réaliser en un temps fixe). Ce qu'il faut retenir, c'est que les algorithmes polynomiaux sont les seuls à pouvoir être utilisés informatiquement pour de grandes valeurs de n, et ce, quelle que soit la puissance de la machine.

**Définition :**

- On dit qu'un problème est dans ***P*** s'il existe un algorithme pour le résoudre en temps *polynomial*.
- On dit qu'un problème est dans **NP** s'il existe un algorithme pour vérifier qu'une solution donnée convient en un temps *polynomial*.

La définition d'un problème NP peut sembler complexe. Essayons de l'éclaircir. Un des problèmes difficiles des mathématiques est la factorisation d'un entier en produit de facteurs premiers. On ne sait pas s'il existe un algorithme polynomial qui réussisse cette opération. Autrement dit, on ne sait pas si ce problème est dans P. En revanche, étant donné des nombres $p_1,...,p_k$, il est trivial de vérifier si $n=p_1...p_k$ : ce problème est dans NP.

Pour résumer, être dans P, c'est *trouver* une solution en temps polynomial, tandis qu'être dans NP, c'est *prouver* en temps polynomial qu'une proposition de réponse est une solution du problème. Ainsi, tout problème qui est dans P se trouve dans NP. Un champ de recherche majeur des mathématiques actuelles est l'investigation de la réciproque : a-t-on ***P=NP?*** Autrement dit, peut-on trouver en temps polynomial ce que l'on peut prouver en temps polynomial? Ce problème est si important qu'il fait partie des 7 problèmes du millénaire, dont la résolution est primée 1 million de dollars par le Clay Mathematic Institute.

**Définition :** On dit qu'un problème est *NP-complet* si la résolution de ce problème en temps polynomial entraîne la résolution en temps polynomial de tout problème NP.

Les problèmes *NP-complets* sont donc les plus compliqués des problèmes NP.

*4.4.3- Programmation linéaire :*

La programmation linéaire consiste à maximiser (ou à *minimiser*) une fonction linéaire sous des contraintes linéaires (ces contraintes sont le plus souvent exprimées par des inégalités). « Linéaire » s'entend ici par « du premier degré ». Elle intervient dans la résolution de problèmes combinatoires.

Exemple :

Objectif : Max (3x + 7y - 2z)

Sous contraintes :

2x + 5y - z <= 15
-3x + 15y + 4z <= 58
x >= 0, y >= 0 et z >= 0

Le terme programmation linéaire suppose que les solutions à trouver doivent être représentées en variables réelles. S'il est nécessaire d'utiliser des variables discrètes dans la modélisation du problème,

on parle alors de programmation linéaire en nombres entiers (PLNE). Il est important de savoir que ces derniers sont nettement plus difficiles à résoudre que les PL à variables continues

### *4.4.4- Programmation linéaire en nombres entiers :*

Un problème de programmation linéaire en nombres entiers (PLNE) est un programme linéaire, c'est-à-dire une fonction objectif linéaire à maximiser ou minimiser, sous contraintes linéaires, dans lequel il y a la contrainte supplémentaire que les variables sont entières. On parle de **programme linéaire mixte** lorsque seul un sous-ensemble de variables doivent être entières et les autres réelles.

Il est facile de montrer que la PLNE est un problème *NP-complet* car de nombreux problèmes NP-complets peuvent être exprimés comme des PLNE. Pour les résoudre, la relaxation continue d'un PLNE (c'est le PLNE sans les contraintes d'intégrité) est un PL qui peut être résolu efficacement. Les algorithmes de résolution de PLNE, tels que les algorithmes par *séparation et évaluation* ou les algorithmes de génération de *plans sécants*, se basent très souvent sur cette relaxation continue.

On a déjà dit, dans la section de plus court chemin, qu'il existe d'autres manières pour calculer les plus court chemin : en effet, une des façons existantes utilise le problème des flots à présenter dans la section qui suit..

## ***4.5- Problèmes des flots sur un graphe :***

Les flots permettent de modéliser une très large classe de problèmes. Leur interprétation correspond à la circulation de flux physiques sur un réseau : distribution électrique, réseau d'adduction, acheminement de paquets sur Internet, ... Il s'agit d'acheminer la plus grande quantité possible de matière entre une source ***s*** et une destination ***t***. Les liens permettant d'acheminer les flux ont une capacité limitée, et il n'y a ni perte ni création de matière lors de l'acheminement : pour chaque noeud intermédiaire du réseau, le flux entrant (ce qui arrive) doit être égal au flux sortant (ce qui repart).

### *4.5.1- Définition du flot et du multiflot sur un graphe (réseau):*

Un **réseau** est un graphe orienté ***G=(V,A)*** avec une valuation positive de ses arcs. La valuation ***c(x,y)*** d'un arc ***(x,y)*** est appelée la **capacité** de l'arc. On distingue sur ***V*** deux sommets particuliers : une source ***s*** et une destination ***t***. Les autres sommets sont les noeuds intermédiaires du réseau.

Un flot représente l'acheminement d'un flux de matière depuis une source ***s*** vers une destination ***t***. Le flot est ainsi décrit par la quantité de matière transitant sur chacun des arcs du réseau. Cette quantité doit être inférieure à la capacité de l'arc, qui limite ainsi le flux pouvant transiter par lui. De plus il n'est pas possible de stocker ou de produire de la matière aux noeud intermédiaires : un flot vérifie localement une loi de conservation analogue aux lois de ***Kirshoff*** en électricité.

Un **flot** ***F*** sur un réseau ***G=(V,A)*** est une valuation positive des arcs, c'est à dire une application de ***A*** dans ***R+***, qui vérifie les deux propriétés suivantes :

1. Pour tout arc $a \in A,\ 0 \le F(a) \le c(a)$
2. Pour tout sommet intermédiaire $x \in \mathrm{V} \setminus \{\mathrm{s},\mathrm{t}\},\ \sum_{y} F(y,x) = \sum_{y} F(x,y).$

Les problèmes des flots sont des problèmes fondamentaux et très souvent utiles dès qu'il s'agit de trouver des chemins dans les graphes. Ils se départagent en deux catégories, le simple ***flot*** et le ***multiflot.***

Un ***multiflot*** consiste à faire cohabiter plusieurs flots sur le réseau de sorte que la somme des flots passant sur un arc soit inférieure à la capacité. Et ce problème est utile dans un grand nombre d'applications, notamment lorsqu'il est question de calculer des routes pour des entités qui sont en concurrence pour certaines ressources, ressources qui sont modélisées par des capacités sur les arcs du graphe support des routages possibles.

*4.5.2- Problème de minimisation du coût du flot circulant sur un graphe :*

Le problème de minimisation de coût du flot est le problème le plus fondamental au niveau de tous les problèmes de flots dans les réseaux.

Soit ***G=(V,A)*** un réseau orienté définie par un ensemble de nœuds « ***V*** » et « ***A*** » définit un ensemble de ***m*** arcs. $\forall arc(i,j) \in A$ un coût $c_{ij}$ est associé pour définir le coût par unité de flot sur cet arc. On assume que le coût du flot varie linéairement avec la quantité du flot. On associe avec chaque arc $(i,j) \in A$ une capacité $u_{ij}$ qui définie la quantité maximale de flot qui peut circuler sur cet arc. On associe avec chaque nœud $i \in V$ un nombre entier $b(i)$ représentant la production ou la consommation (en terme de flot) de ce nœud. Si $b(i) \succ 0$, le nœud « i » est un noeud producteur ; si $b(i) \prec 0$, le nœud « i » est un noeud consommateur ; et si $b(i) = 0$, le nœud « i » est un noeud intermédiaire servant de relais. Les variables de décision dans ce problème sont les flots sur les arcs et on représente le flot sur l'arc $(i,j) \in A$ par $x_{ij}$. Le problème de minimisation de coût du flot est un problème d'optimisation formulé comme suit :

Objectif : Minimiser $\sum_{(i,j)\in A} c_{ij} x_{ij}$

Sous contraintes :

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \qquad \forall i \in V \qquad (1)$$

$$0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in A \qquad (2)$$

Au niveau de la contrainte n°1, le premier terme désigne le flot sortant du noeud « i » et le second terme désigne le flot entrant au nœud « i ». !cette contrainte est connu sous le nom de la contrainte de conservation de flot. Tandis que la contrainte n ° 2 est connue sous le nom de contrainte de capacité, et elle assure que le flot circulant sur un arc ne dépasse pas la capacité disponible sur ce dernier.

Le problème présenté peut servir à résoudre le problème de plus court chemin entre une source $s \in V$ et une destination $t \in V$ . En effet, il suffit de mettre :

- $b(s) = 1.$
- $b(t) = -1.$
- $b(i) = 0 \quad \forall i \in V \setminus \{s,t\}.$

Ainsi la solution enverra une seule unité de flot de la source « *s* » vers la destination « *t* » par le plus court chemin.

# Chapitre 5

# PROBLEMATIQUE ET CAHIER DE CHARGE

La problématique qui nous intéresse peut être simplement décrite par la question suivante :

- Comment assurer la meilleure utilisation d'une infrastructure existante face à un trafic dynamique ?

Dans le cadre de notre travail, on va utiliser le routage adaptatif pour la mise en place des connexions générées. La décision concernant l'acceptation ou le rejet d'une demande de connexion est prise par un nœud central qui, en se basant sur l'état actuel du réseau et pour chaque demande de connexion, va essayer de trouver un chemin valide ; Ce problème peut être traité de plusieurs manières différentes :

- Le nœud central cherche à trouver des chemins en se basant seulement sur l'état actuel du réseau.
- Le nœud central cherche à trouver des chemins en se basant sur l'état actuel du réseau et en sachant les « ***k*** » demandes de connexion futures. (k est un entier quelconque).
- Le nœud central cherche à trouver des chemins en se basant sur l'état actuel du réseau et en sachant toutes les demandes de connexion futures.

Notre objectif ainsi est donc de mettre en place une solution permettant de trouver des chemins valides à des demandes de connexions générées à une étape tout en tenant compte de ce qui parviendra au niveau du futur. On a appelé, cette approche « ***routage avec prise en compte du futur*** » et notre but est de l'implémenter, l'évaluer et de la comparer à une stratégie utilisant le routage ***myope*** (routage des demandes dès leurs arrivées).

Globalement, la tâche finale proposée peut être divisée en 5 étapes élémentaires :

- La génération de scénarios de trafic pour les simulations tout en assurant leurs persistances.
- Proposition et implémentation d'un algorithme de routage à l'aveugle.
- Proposition et implémentation d'un algorithme de routage avec prise en compte du futur.
- L'évaluation de performance des 2 algorithmes en procédant à des simulations objectives.

Il est à noter que toutes les solutions algorithmiques proposées doivent être implémenté au niveau de la bibliothèque d'optimisation combinatoire ***Mascopt*** que je présenterais au niveau du chapitre suivant.

# Chapitre 6

# PRESENTATION DE LA PLATEFORME DE TRAVAIL

Tous mes travaux ont été programmés sous le langage orienté objet Java, et plus précisément implémentés au niveau de la bibliothèque d'optimisation ***Mascopt***. Pour la résolution des programmes linéaires proposés le solveur ***CPLEX*** a été utilisé. Dans ce qui suivra, nous présenterons une brève description de la bibliothèque d'optimisation ***MASCOPT*** et le solveur ***CPLEX***.

## ***6.1- Mascopt :***

Avant de présenter ***Mascopt***, il sera plus intéressent d'exposer les causes de développement d'une telle bibliothèque d'optimisation libre. En effet, il est à noter qu'avant de commencer un tel projet, Mascotte était en train de travailler sur le projet RNRT avec ses partenaires ALCATEL SPACE RESEARCH & INNOVATION et FRANCE TELECOM sur le développement du logiciel PORTO (pour **P**lanification et **O**ptimisation des **R**éseaux et de **T**ransport **O**ptiques). Ce logiciel concrétise la vision d'un opérateur qui souhaite allouer pour ses clients les ressources de son réseau WDM : il aura en entrée un réseau physique (nœuds et câbles) et une matrice de trafic à réaliser (exprimés en trames STM fournie par FRANCE TELECOM) et les modules d'optimisation calculent l'allocation des conteneurs optiques (fibres, bandes ou longueurs d'onde) en minimisant une fonction de coût associée au nœuds du réseau (donnée ici par ALCATEL). Ainsi, PORTO n'était pas, à proprement parler, une bibliothèque d'optimisation mais plutôt une application dédiée à l'optimisation des ressources des réseaux optiques. Aussi, le développement fut axé sur le développement d'algorithmes performants répondants à des problèmes posés par ALCATEL SPACE RESEARCH & INNOVATION et FRANCE TELECOM ainsi que sur le développement d'une interface 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 les développeurs n'ayant pas connaissance des choix d'implémentations de PORTO, de s'approprier l code pour en faire un usage connexe. Un autre problème, gênant les personnes travaillant sur les sujets de recherche, consiste à ne pas pouvoir reproduire l'exécution d'algorithmes trouvés dans les publications et les rapports de recherche à cause de manque d'informations concernant la manière d'implémentation. Afin de dépasser ces problèmes, MASCOPT a été développé.

***Mascopt*** (contraction de Mascotte et d'optimisation), profitant de l'expérience de PORTO, est une bibliothèque Java dédiée aux problèmes d'optimisation dans les réseaux, notamment pour l'écriture de prototypes logiciels. Cette open source offre l'implémentation d'un

modèle générique de graphe. Elle a été conçue dans l'esprit des modèles orientés objet et privilégie l'accessibilité et la facilité d'utilisation à la vitesse d'exécution des programmes.

Nous montrons aussi comment ce modèle de graphes peut être étendu et spécialisé à des cas d'utilisation particuliers en utilisant des mécanismes objets simples. Enfin, une description rapide des fonctionnalités de *MASCOPT* est présentée.

La modélisation des graphes dans cette librairie se fonde sur la théorie des graphes classique. Un graphe G est donc une paire $(V, E)$ où $V = \{v_1, v_2, ..., v_n\}$ est l'ensemble de sommets et $E = \{e_1, e_2, ..., e_m\}$ l'ensemble des arêtes. Chaque élément de V ou de E doit être instancier par l'utilisateur qui a la possibilité de les manipuler librement. De ce fait, il est essentiel que V et E soient cohérents et se correspondent c'est-à-dire que tous les sommets des arcs de V appartiennent à E, pour que le graphe G soit valide.

Cette modélisation utilise intensivement les possibilités de gestion des objets. Dans le sens où aucun objet n'est dédié à un graphe en particulier mais peut être utilisé pour plusieurs graphes ou éléments de graphes. Par exemple, on peut instancier deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V1, E_2)$ partageant le même ensemble de sommets. Ainsi les deux graphes se différencient seulement par leur ensemble d'arêtes et l'ensemble de sommets n'est pas doublé en mémoire.

Pour chaque type d'élément (graphe, sommet, arc...), les attributs et les méthodes communs sont groupés dans des classes abstraites. De cette façon, la bibliothèque profite pleinement du principe de transmission : la méthode de construction de graphe donne la possibilité à l'utilisateur d'écrire ses propres classes de sommet et d'arcs en les dérivant des classes de *Mascopt*.

Ainsi, l'utilisateur peut utiliser à son aise les algorithmes de la bibliothèque sur son modèle de graphes ; ce qui évite une duplication du code et une meilleure gestion de la mémoire.

*Mascopt* est organisée suivant une architecture en huit packages. Chaque package fournit plusieurs interfaces menant les utilisateurs à construire des classes similaires adaptées à leurs besoins.

- **mascoptLib.abstractGraph :** est le package de la librairie abstraite de graphe. Y sont implémentés toutes les fonctionnalités des classes du schéma ci-après.
- **mascoptLib.algos.abstractalgos :** est un package qui fournit les principaux algorithmes génériques pour la classe *AbstractGraph* **,** ainsi que pour d'autres objets du package.
- **mascoptLib.graphs :** est un package dédié aux classes *Graph* et *DiGraph*.
- **mascopt.gui :** est une interface graphique pour l'utilisateur. Ce package est la couche graphique du package *mascoptLib.abstractGraph*.
- **mascoptLib.io.graph :** est un package qui implémente toutes les fonctionnalités des entrées/sorties possibles. Plusieurs formats sont fournis pour enregistrer dans un fichier les objets des différents packages.
- **mascoptLib.io.util :** est un package contenant les outils nécessaires pour manipuler les fichiers écrits par le package *mascoptLib.io.graph*.
- **mascoptLib.util :** est un package qui contient les objets et les algorithmes qui ne sont pas dédiés aux graphes.

Nous détaillons brièvement le fonctionnement et les interactions des principales classes du package *mascoptLib.graphs* afin de simplifier la lecture et de comprendre les bases de cette bibliothèque.

- *Les objets simples :*

  - ***MascoptObject*** met en place un système d'enregistrement de valeur (un *Double* ou un *Integer* ou encore un *String*) sur l'objet c'est à dire qu'il est possible de faire pointer sur un objet *MascoptObject* des objets tel que un Double ou un *Integer* ou encore un *String*. Ces valeurs sont accessibles et modifiables par le biais de fonctions (*getValue(), setValue(),...*).

  - ***Vertex*** est l'objet le plus fondamental qui peut être construit. L'objet *Vertex* fournit des informations sur ses voisins, sur ses degrés, sur ses arcs entrant ou sortant. Il dérive des classes *AbstractVertex* et de *MascoptObject*.

  - ***Edge*** et ***Arc*** : l'objet *Edge* se construit à partir de deux *Vertex*. Cet objet fournit, à partir d'un sommet donné, des méthodes qui renvoie le sommet opposé à celui-ci ou qui teste si l'arc mène à ce sommet,... l'objet *Edge* dérive des classes *AbstractEdge* et *MascoptObject*. La classe Arc dérive de *AbstractEdge* et implémente des arcs orientés.

- *Les ensembles :*

  - ***MascoptSet*** est une classe dérivée de *MascoptObject*. Cette classe met en place des interfaces pour les ensembles ou les collections du package. Elle offre une implémentation efficace ayant les mêmes caractéristiques que la classe *HashSet* (représentant une table de hachage dans le langage Java). De plus, des fonctionnalités supplémentaires y sont ajoutées pour pouvoir faire pointer des objets tel que un *Double* ou un *Integer* ou encore un String directement sur les objets des ensembles.

  - La classe ***VertexSet*** dérive de la classe ***AbstractVertexSet*** et de *MascoptSet*. Elle permet de regrouper un ensemble d'objet *Vertex*.

  - ***EdgeSet*** et ***ArcSet*** : les classes *EdgeSet* et *ArcSet* dérivent des classes ***AbstractEdgeSet*** et de *MascoptSet*. Elles permettent de former un ensemble d'objets *Edge*, respectivement d'objets *Arc*.

- *Graphe orienté et non orienté :*

  - La classe ***Graph*** construit un graphe non orienté avec un ensemble de sommets et d'un ensemble d'arêtes. Cette classe garantie qu'à tout moment les objets *EdgeSet* et *VertexSet* coïncident et sont cohérents. Elle offre de manière

très simple la possibilité de pouvoir copier des graphes et de construire des sous graphes.

- La classe ***DiGraph*** construit de la même manière un graphe orienté avec les objets *ArcSet* et *VertexSet*.

Ces deux classes dérivent de ***AbstractGraph.***

- *Chemin orienté et non orienté :*

  - L'objet ***Path*** permet de construire un chemin au-dessus d'un graphe. Un chemin peut-être fusionné avec un autre chemin en gardant le même sommet de début et de fin, donnant un chemin multiple. Le chemin est considéré comme un graphe et dérive de l'objet *Graph*.

  - L'objet ***DiPath*** est un chemin orienté. Il dérive de l'objet *DiGraph*.

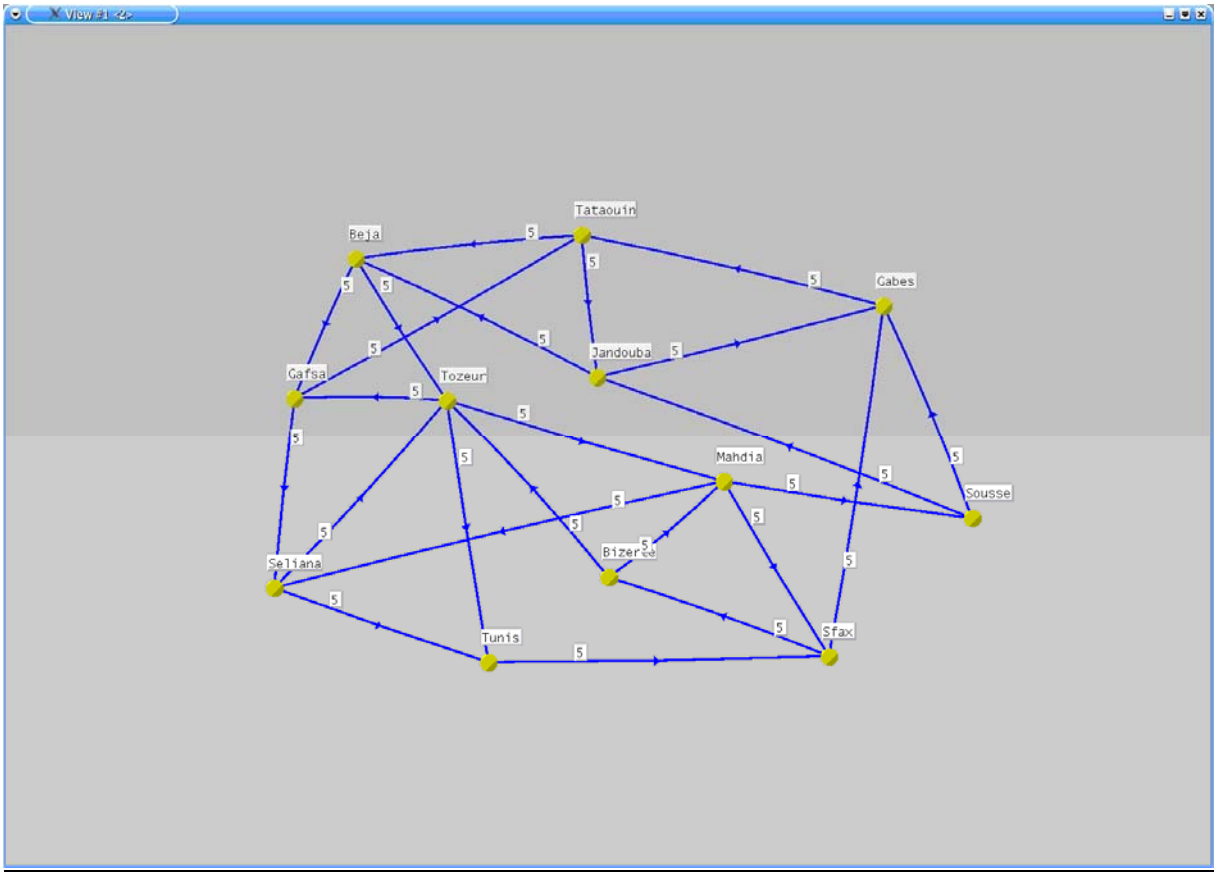



**Figure 6.1 : exemple de graphe dessiné avec Mascopt**

## ***6.2- ILOG CPLEX :***

*ILOG CPLEX* est un ensemble d'algorithmes permettant de résoudre différents problèmes allant de la programmation linéaire en nombres fractionnaires à la programmation linéaire en nombres entiers. Il peut être utilisé sous plusieurs formes, principalement sous la forme d'un logiciel en lignes de commandes et sous la forme d'une librairie C/C++ ou JAVA. Celle que l'utilise sous l'environnement JAVA, est la librairie de classes « *ILOG Concert Technology for the Java platform* » offrant une API avec des facilités de modélisation qu'on peut utiliser pour embarquer les optimisateurs CPLEX au niveau des applications Java.

Le package qui m'était le plus utile durant mon travail est le package :

- *Ilog.cplex.ILOCplex* : ce package permet de définir un programme linéaire, de lui ajouter les variables, définir les contraintes et finalement le résoudre. Une résolution peut être faite si le problème admet une ou plusieurs solutions sinon la résolution échoue. L'extraction des valeurs des variables suite à une résolution n'est pas si difficile : en effet il suffit juste de sauvegarder les variables qu'utilise *Cplex* dans une structure de donné de l'application JAVA et le package définit des méthodes simple pour avoir les valeurs de ces variables.

Il à noter que ILOG CPLEX est un logiciel propriétaire et il faut acquérir une licence pour pouvoir l'utiliser.

# Chapitre 7

# TRAVAIL REALISE

Le travail réalisé consiste en 4 grandes parties à savoir :

1- La génération des scénarios de trafic.
2- Le routage Myope
3- Le routage Avec Prise en compte du futur.
4- Les expérimentations et les résultats

Comme vous allez le remarquer lors de la lecture de cette partie, cette décomposition n'est pas finale : en effet suite aux résultats trouvés au niveau des expérimentations d'autres parties ont été ajoutées.

## ***7.1- Génération des scénarios de trafic :***

En premier lieu, et comme il a été mentionné au niveau du cahier de charge, il faut mettre en place une application permettant :

- La génération de scénarios de trafic pour réaliser les simulations.
- D'assurer la persistance des scénarios de trafic produits (à fin de permettre leurs réutilisations).

Lors de la construction des scénarios de trafic, on suivra évidemment certaines caractéristiques et en particulier au niveau des temps de séjours relatifs aux demandes de connexion. L'objectif est d'être capable de fixer moyennement la charge produite sur le réseau à chaque étape.

### *7.1.1- Format de demande de connexion :*

Pour notre travail, une demande de connexion est décrite comme suit:

| *@source* | *@destination* | *Etape_arrivée* | *Temps_séjour* | *Capacité_réquise* |
|---|---|---|---|---|

- *@source* : origine de la demande de connexion.
- *@destination* : destination de la demande de connexion.
- *Etape_arrivée* : l'étape d'arrivée de la demande de connexion.
- *Temps_séjour* : ensemble d'étapes pendant les quels la connexion restera sur le réseau.

- *Capacité_réquise* : les unités de capacités requises par la demande au niveau des ressources du réseau.

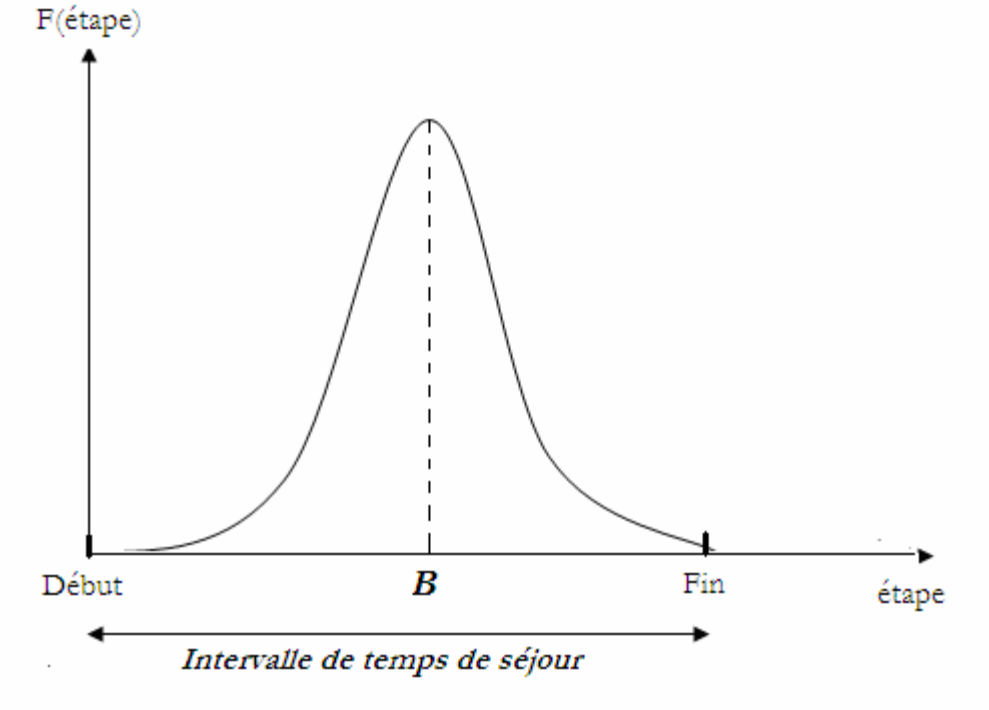
*7.1.2- Modèle de génération des demandes de connexion :*

Au niveau de l'***@source***, l'***@destination***, ***Etape_arrivée*** et ***Capacité_réquise*** leurs génération suivra une distribution uniforme sans oublier que l'@source devra être différente de l'@destination et la ***Capacité_réquise*** strictement supérieure à 0.

Vue que nous traitons un cas de réseau fonctionnant en mode circuit, ce qui nous intéresse le plus sera le temps de séjour d'une connexion sur le réseau : en d'autres termes ça reviendra à exprimer durant combien d'étapes des ressources vont être utilisées par des connexions.

L'idée de bas était d'essayer de générer un trafic avec un temps de séjour moyen égal à $\beta$ étapes (qu'on fixera bien sûr suivant nos besoins en vue d'assurer la charge moyenne désirée sur le réseau à chaque étape). Partant de ce principe, une distribution de nombres de demandes de connexion générées à chaque étape suivra la fonction suivante :

$$F : etape \rightarrow etap * e^{(-etape/\beta)}$$



Pour spécifier la valeur moyenne autour de laquelle vont prendre valeurs les temps de séjour de l'ensemble des requêtes, la distribution précédente a été présenté et notre but est de générer :

- Beaucoup de demandes de connexion qui vont établir des connexions ayant à rester sur le réseau pendant des périodes proches de la valeur moyenne.
- Peu de demandes de connexion qui vont établir des connexions ayant à rester pendant des périodes assez éloignées de la valeur moyenne (dans notre cas peu de requêtes qui vont rester peu de temps ou tout le temps).

A fin de suivre une telle distribution une solution simple est décrite comme suit (mais il est à signaler qu'on va procéder à déterminer en premier lieu, pour chaque étape le nombre de demandes de connexion qui vont prendre la valeur de cette étape comme étant leurs temps de séjour sur le réseau) :

1. Pour chacune des étapes contenues dans l'intervalle de temps de séjour, calculer $F(étape)$ et l'ajouter à une valeur « ***somme ».***
2. Ainsi « $F(etape)/somme$ » désignera le pourcentage des demandes de connexion à générer et qui auront comme temps de séjour une valeur égale à « ***étape »***.
3. Pour savoir comment attribuer à « ***x*** » (entier quelconque) demandes de connexion leurs temps de séjour on procédera comme suit :

   a - Pour chaque étape « ***k*** » (dans un intervalle donné de temps de séjour) : le nombre de requêtes (requêtes de demandes de connexion) ***nbr_req(k)*** qui vont rester ***k*** étapes sera égal à (bien sûr avec quelques normalisations):

   $$nbr\_req(k) = ((F(etape)/somme) * x)$$

   b - Créer pour chaque temps de séjour l'ensemble des requêtes relatives.

Après il sera facile d'attribuer à chacune des requêtes produites une '***@source***, une ***@destination***, une ***Etape_arrivée*** et une ***Capacité_réquise*** tirées aléatoirement de leurs intervalles relatifs.

Il est à remarquer que pour faire mieux, on a crée l'ensemble des requêtes avec leurs périodes de séjour, ensuite on les a mélangé et en fin on leurs a attribué le reste des paramètres tirés d'une manière aléatoire.

*7.1.3- Persistance des scénarios générés:*

Pour assurer la persistance des scénarios générés, on a préféré sauvegarder ces derniers dans des fichiers ASCII simples au lieu des fichiers XML (une solution déjà utilisé au niveau de ***MASCOPT***) et ce pour une simple raison décrite comme suit :

- La sauvegarde des scénarios de trafic ou leurs chargements des fichiers XML prend un temps assez grand relativement au temps mis pour effectuer les mêmes opérations

en utilisant des fichiers ASCII et ce à cause des tailles assez grande des scénarios qu'on estime générer.

## ***7.2- Routage myope :***

### *7.2.1- Problème d'attribution de chemin :*

L' « *Attribution de chemin* » est le fait d'assigner à une demande de connexion point à point un chemin entre la source et la destination.

Un problème d'attribution de chemin est dit faisable si les deux critères suivants sont satisfaits :

1- Tout le trafic correspondant à une demande est routé à travers un des chemins disponibles.
2- Le volume total du trafic routé à travers un lien quelconque ne doit pas dépasser la capacité disponible sur ce lien.

La phase primordiale d'un problème d'attribution de chemin est celle de recherche du chemin optimal (le plus court chemin) en se basant sur les coûts associés à chaque lien du réseau : en effet le but est de choisir pour une demande quelconque le chemin optimal de la liste des chemins disponibles pour cette demande.

### *7.2.2- Définition du routage myope :*

Le routage myope revient à trouver simplement pour chaque demande de connexion arrivante au réseau le plus court chemin en terme de coût sans se soucier de ce qui arrivera dans les prochaines étapes.

Il est à noter que le temps considéré dans la résolution de notre problème est un temps discret et pas continu, ainsi une étape sera la période de temps suffisante pour :

- Calculer le chemin adéquat pour les demandes de connexion générées sur le réseau à une étape « ***t*** ».
- Réaliser les mises à jour nécessaires suite à la mise en place ou le retrait des demandes de connexion ou des connexions du réseau.

### *7.2.3- Algorithme formel proposé :*

L'algorithme formel proposé assurant ce fonctionnement est le suivant :

**Algorithme1** Algorithme du routage myope

**Entrée :** $G = (V, E)$, *Duree_simulation*, $D = \{d_1, d_2, ..., d_m\}$ (ensemble des demandes de connexions triées dans un ordre croissant suivant leurs temps d'arrivée}, *nombre_demandes*.
**Sortie :** Taux de réussite
**Variables locales :**

*connexions_finies_etape* : vecteur servant à stocker les connexion finies à une étape spécifiée.
*demandes_entrantes_etape* : vecteur servant à stocker les demandes de connexion générées à une étape spécifiée.
*Taux_réussite* : Pourcentage des demandes acceptées sur le réseau.
*Chemin* : chemin trouvé pour une demande de connexion spécifiée
*demandes_rejetées* : nombre de connexions rejetées tout au long de la simulation.

**1. Pour** chaque « etape » **allant de** 1 **à** *Duree_simulation* **faire**
**2.** Vider *requetes_sortantes_etape*
**3.** *connexions_finies_etape* ← *trouver_requêtes_sortantes*(*etape*)
**4.** **Si (***connexions_finies_etape* est non vide) **faire**
**5.** **Pour** chaque « *connexion* » de l'ensemble *connexions_finies_etape* **faire**
**6.** *Mettre_à_jour_capacités_et_coûts_des_liens_du_chemin_de*(*connexion*)
**7.** **Fin Pour**
**8.** **Fin Si**
**9.** Vider *demandes_entrantes_etape*
**10.** *demandes_entrantes_etape* ← *trouver_demandes_entrantes*(*etape*)
**11.** **Si (***demandes_entrantes_etape* est non vide) **faire**
**12.** **Pour** chaque « *demande* » de l'ensemble *demandes_entrantes_etape* **faire**
**13.** *Mettre_à_jour_coût_des_liens_suivant_la_capacité_requise_par*(*demande*)
**14.** *Chemin* ← *Plus_court_chemin*(*demande*)
**15.** **Si** ***(*** *Chemin est non vide* ***)*** **Faire**
**16.** *Enregistrer_demande_chemin(demande,chemin)*
**17.** *Mettre_à_jour_capacités_et_coûts_des liens_composant*(*Chemin*)
**18.** **Sinon** *demandes_rejetées* ← *demandes_rejetées + 1*
**19.** **Fin Si**
**20.** **Fin Pour**
**21.** **Fin Si**
**22. Fin Pour**
**23.** *taux réussite* $\leftarrow \dfrac{(nombre\_demandes - demandes\_rejetées)}{nombres\_demandes} * 100$
**24. retourner**(*taux réussite*)

*7.2.4- Description détaillée de l'algorithme proposé :*

Après avoir extrait le scénario des demandes de connexions, un certain nombre de traitements doivent être appliqués avant de se soucier de la recherche du plus court chemin et d'autres traitement après avoir trouver ce plus court chemin , et en voici une description :

- *Mettre à jour les capacités et les coûts sur les liens suite à la sortie des requêtes :*

Suite à la fin d'une ou plusieurs connexions, des unités de capacité vont être évidemment libérées sur les liens déjà empruntés par la (les) requête(s). Suite à ça, une opération de mise à jour de coût et de capacité est exigée au niveau de ces liens.

- *Mettre à jour les coûts sur les liens suivant la capacité requise par une requête :*

Cette phase est une phase intermédiaire permettant l'élimination des liens, n'assurant pas la capacité requise par une demande de connexion, de la liste des liens ensemble de recherche du plus court chemin. (Mettre un coût très grand pour les liens dont la capacité disponible est inférieure à la capacité requise par la demande).
Il est à noter que les fonctions assurant le calcul des coûts vont être traité en détail plus tard.

- *Trouver le plus court chemin pour la demande :*

Au niveau de cette étape, l'algorithme de ***DIJKSTRA***, utilisé pour le calcul du plus court chemin, va être appliqué sur l'instance actuelle du réseau (Le réseau avec ses valeurs actuelles des coûts sur les liens).

- *Mettre à jour les capacités et les coûts sur les liens :*

Après avoir trouver le chemin le moins coûteux sur le réseau, une tâche de mise à jour des capacités et des coûts sur les différents liens parcourus doit être déclenchée. En fait la mise à jour des capacités consiste en la soustraction de la capacité requise par une requête des capacités disponibles sur l'ensemble des liens parcourues. Par contre la mise à jour des coûts n'est pas si évidente. En effet un nouveau coût du lien est calculé à base de la nouvelle capacité disponible. D'où la nécessité de présenter quelques fonctions de coûts fréquemment utilisées.

### *7.2.5- Fonctions de coût proposées :*

Généralement les fonctions du coût sont formulées et utilisées pour mettre en œuvre une stratégie d'attribution de ressources. Deux stratégies principales peuvent être distinguées :

- PACK strategy.
- SPREAD strategy.

#### *7.2.5.1- PACK strategy :*

Cette stratégie essaye d'emballer les demandes de connexion dans un nombre minime de liens, les demandes sont routées à travers les plus courts chemins ce qui revient à utiliser de plus en plus les liens les plus saturés.

On peut dire aussi que cette stratégie essaye de router les requêtes par les plus courts chemins en termes de sauts tant que c'est possible (il reste encore de la capacité sur les liens).

La fonction du coût la plus simple est définit comme suit :

- $cout(e) = cste$ si il reste encore de la capacité disponible sur le lien.
- $cout(e) = +\infty$ s'il ne reste plus de capacité disponible sur le lien.

La constante peut bien être calculé en fonction de la longueur physique du lien, le coût d'installation du lien, etc. (le plus important ici est qu'elle ne change pas de valeur).

*7.2.5.2 - SPREAD strategy :*

Ici, dépendante de la fonction de coût, la stratégie peut distribuer les demandes sur les liens En effet on ne route plus la demande par le plus court chemin à base de nombre de sauts mais par celui qui nous coûte le moins cher en terme de congestion : toutes les fonctions qui vont être présentées se basent sur le principe de pénalisation suite à l'utilisation de la capacité sur les liens (ce qui revient à utiliser tant que c'est possible les chemins les moins congestionnés).

La fonction du coût est définit comme suivant :

- $cout(e) = f(capacité\_utilisée)$ si il reste encore de la capacité disponible sur le lien.
- $cout(e) = +\infty$ s'il ne reste plus de capacité disponible sur le lien.

La fonction « $f(capacité\_utilisée)$ » est une fonction de coût croissante dont la vitesse de croissance peut suivre une exponentielle, une parabole, ou même une discrétisation de l'intervalle de la capacité utilisée.

*Exemples de fonctions de coût :*

Dans cette section on va présenter quelques fonctions d'une façon ascendante par rapport à leur vitesse de croissance pour les calculs du coût :

Pour l'ensemble des fonctions qu'on va présenter les notations suivantes vont être utilisées :

- *capacite* : capacité totale sur un lien.
- *capacite_restante* : capacité restante sur un lien.
- *capacite_utilisee* : capacité utilisée sur un lien.

Une égalité bien évidente serait :

$$capacite = capacite\_restante + capacite\_utilisee.$$

*Fonctions proposées :*

- Fonction à croissance exponentielle :

$$f : capacite\_utilisee \rightarrow e^{capoacite\_utilisee}$$

- Fonction à croissance parabolique :

$$f : capacite\_utilisee \rightarrow capacite\_utilisee^2$$

- Fonction discrétisée :

Au niveau de cette fonction une discrétisation de l'intervalle de capacité sur un lien est requise et après un coût bien défini devrait être attribué à chaque portion de l'intervalle discrétisé.

La figure suivante exprime la capacité utilisée sur une arête quelconque.

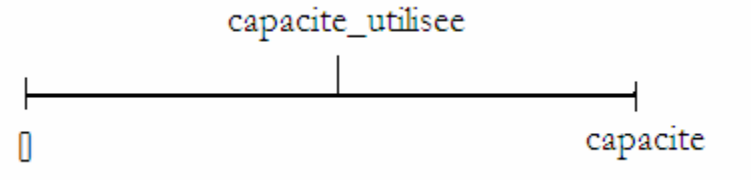



**Figure : capacité utilisée sur un lien**

Le paramètre de congestion se définie comme suit :

$congestion = capacite\_utilisee / capacite$ (Varie clairement entre 0 et 1).

On pourra par exemple définir une fonction discrète en fonction de la congestion.

*Exemple :*

Un exemple identifiant la mise à jour d'un coût sur un lien « ***e*** » suivant la valeur de sa capacité disponible.

Si *congestion* ≤ 0,5 alors *cout(e)* = 1
Sinon si 0,5 < *congestion* ≤ 0,8 alors *cout(e)* = 10

Sinon $cout(e) = 1000$

### ***7.3- Routage avec prise en compte du futur, 1er Modèle proposé :***

L'idée de base peut être exposé à travers la question suivante :

- Si on connaissait le Futur, pourrait-on avoir un routage meilleur que le routage myope ?

La réponse à cette question sera déduite suite aux résultats des simulations qu'on va réaliser.

Router avec prise en compte du futur est le fait de router (trouver un chemin valide) une demande de connexion quelconque tout en tenant compte des demandes de connexion qui vont venir dans les « ***k*** » étapes qui vont suivre.

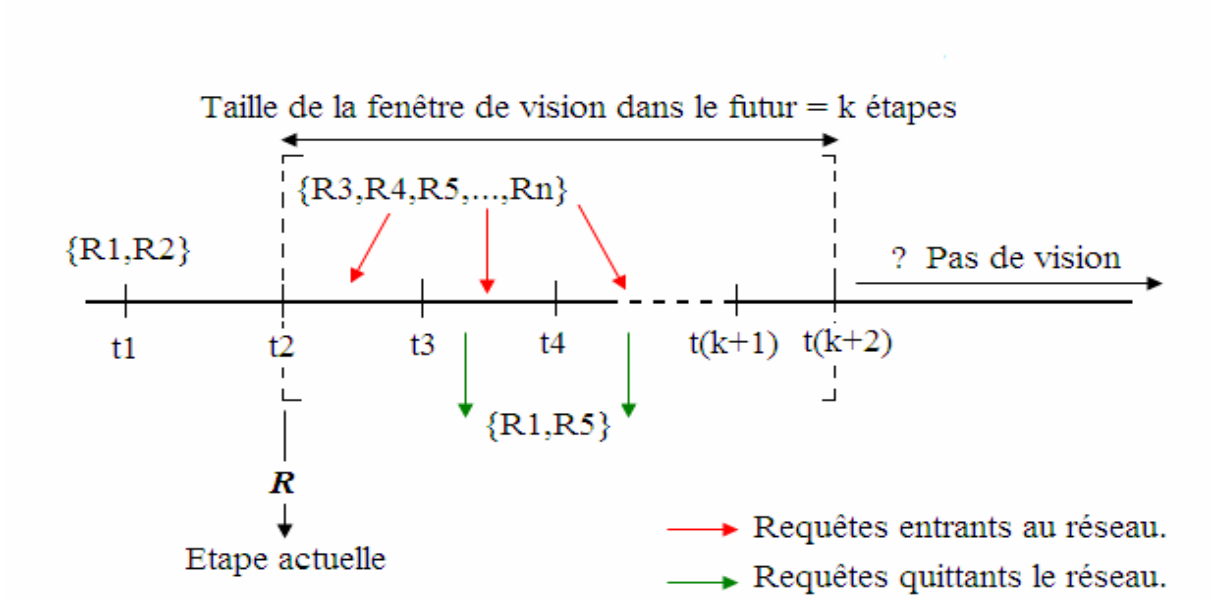



**Figure : Routage avec prise en compte du futur**

Donc d'après la figure précédente, on est à l'étape « t2 » et on veut router la demande de connexion ***R*** en tenant compte des « ***k*** » étapes suivantes : au niveau de ces dernières étapes on aura sûrement des modifications dans l'état de notre réseau suite à l'arrivée de nouvelles demandes ou la sortie d'anciennes demandes déjà routées.

Ainsi, au niveau de cette étape, on se confronte à un problème de routage de plusieurs requêtes (résultantes de l'étape actuelle et des « ***k*** » étapes suivantes) à fin de pouvoir assurer une distribution intelligente des demandes de connexion arrivantes à l'étape *t2* (qui prend en considération le futur).

Un tel problème peut être représenté à travers le problème classique de *Multiflots* mais avec quelques adaptations pour prendre en considération la dynamique du trafic sur le réseau et la progression dans le temps (entrée et sortie des requêtes). Le problème de *Multiflots* est un problème facile s'il est représenté à base de *Programme linéaire à variable fractionnaires* mais devient *NP-COMPLET* si le programme est à base de *variables entières* (comme dans notre cas).

Notre objectif, sera de trouver des chemins valides aux demandes de connexion générées à une étape donnée tout en tenant compte du futur.

Il reste à noter que la correspondance entre ce problème et celui du « *Multiflot* » vient du fait que chaque demande de connexion peut être assimilée à un flot et un « *Multiflot* » n'est rien qu'un ensemble de flots (donc problème de *Multiflot* à chaque étape).

### *7.3.1- Premier modèle proposé : minimisation du coût global du routage :*

#### *7.3.1.1- Description du 1er modèle proposé :*

Pour ce problème, on considère un graphe (réseau) ***G*** : ***G = (V,A)*** constitué de noeuds ***V*** et d'un ensemble d'arcs ***A*** . Sur ce réseau, on considérera un ***coût*** et une ***capacité*** relatifs à chacun des arcs (La capacité peut varier évidemment d'un arc à un autre).

Sur un tel graphe, des scénarios de trafic peuvent être produits suivant des critères bien définis et notre objectif est d'assurer le routage des requêtes qui les composent.

Le modèle utilisé pour formuler le problème est appelé modèle « ***sommet-arc »*** et il nécessite une variable de flot par arc et par requête :

- $T = \{0,1,2,\ldots,T_{max}\}$ : l'ensemble des étapes de la simulation.
- $R$ = l'ensemble des requêtes de demande de connexion du scénario.
- $R_E(t)$= l'ensemble des requêtes générées à une étape t.
- $R_S(t)$= l'ensemble des requêtes sortant du réseau à une étape t.
- $V$ : ensemble de nœuds du réseau.
- $A$ : Ensemble des arcs du réseau.
- *Taille_fenetre* = *k* étapes, avec k : $0,1,2,\ldots, T_{max}$.
- $f_{req}^{a}$ : la variable de décision indiquant la valeur de flot circulant l'arc a et relatif à la requête « req ».
- $C^a$ : coût relatif à l'utilisation de l'arc « ***a »*** durant l'intervalle « ***Taille_fenetre*** ».
- $c^a(t)$ : capacité disponible sur un arc « ***a »*** à l'étape ***t***.
- *pr(v,req)* : le flot sortant (produit) du nœud *v* (v appartient à V) pour la requête *req*.
- *co(v,req)* : le flot rentrant au (consommé par) nœud *v* (v appartient à V) pour la requête *req*.

Nous présentons ci-dessous le programme linéaire relatif à une étape unique. Ainsi, pour résoudre notre problème d'origine ce programme doit être reproduit sur chacune des étapes de la simulation :

*Objectif* : $$\min \sum_{t=T_1}^{T_1+k} \sum_{req \in R(t)} \sum_{a \in A} C^a * f_{req}^a$$

*Sous les contraintes :*

$$\sum_{t=T1}^{t} \sum_{req \in R_E(t)} f_{req}^a \leq c^a(t) + \sum_{i=T_1}^{t} \sum_{req \in R_S(t)} f_{req}^a \qquad \textbf{(1.1)} \text{ avec } T_1 < t \leq T_1 + k$$

$$\sum_{(u,v) \in A} \sum_{req \in R_E(t)} f_{req}^{(u,v)} + pr(v, req) = \sum_{(v,u) \in A} \sum_{req \in R_E(t)} f_{req}^{(v,u)} + co(v, req) \qquad \textbf{(1.2)} \text{ avec } T_1 < t \leq T_1 + k$$

$$\forall req \in R, \forall a \in A, f_{req}^a \geq 0$$

Notre objectif étant la minimisation du coût total du routage, la contrainte n°**1.1** assure que le flot circulant sur un Arc « ***a »*** ne dépasse pas la capacité disponible sur cet arc et la contrainte n°**1.2** sert à assurer la conservation du flot.

Il est à noter que le coût $C^a$ relatif à un arc « ***a »*** est un coût approximatif défini comme suit :

- Pour chaque arc $a \in A$ dont il y'aura libération de la capacité tout au long des étapes de la fenêtre de vision, on se permet d'utiliser le coût calculée avec la valeur de la capacité disponible à la fin de la fenêtre. En d'autres termes, l'idée est tant qu'on sait qu'il y'aura libération de capacité sur un lien donnée dans les étapes futurs on se permet d'utiliser localement ce lien avec un coût moins cher.

Il est à noter aussi que dès que des arcs seront très congestionnés, des requêtes seront rejetées. Pour pouvoir les router, on a recours à relâcher les contraintes de prise en compte du futur en réduisant la taille de la fenêtre, éventuellement à 1. Dès lors, on cherchera d'abord à router les requêtes de l'étape courante. A ce niveau une des deux méthodes suivantes peut être utilisée :

1- Soit router les requêtes une par une en utilisant l'algorithme de recherche du plus court chemin ou algorithme en « ***myope*** ».
2- Soit écrire un programme linéaire permettant de maximiser le nombre de requêtes acceptées tout en tenant comptes du coût engendré par un tel choix.

La deuxième proposition a été retenue et le programme linéaire suivant a été utilisé :

- $R$ : ensemble des requêtes à router.
- $r_{req}$ : variable de décision entière égale à 0 si la requête ne peut pas être router, ou 1 si la requête peut être router et ça pour chaque $req \in R$.

- $A$ : constante à choisir de tel façon à être supérieure à $(B * \sum_{req \in R} \sum_{a \in A} f_{req}^{a})$ (ce choix est pour valoriser plus l'objectif de maximiser le nombre de requêtes à accepter par rapport à la prise en considération du coût.
- $B$ : constante = 1
- $X$ : constante supérieure à $\sum_{req \in R} r_{req}$ car normalement on a à maximiser le nombre de requêtes acceptées et tant qu'on a aussi à minimiser le coût alors on changer le problème en minimisant ($\boldsymbol{X}$ - $\sum_{req \in R} r_{req}$) ce qui revient à minimiser - $\sum_{req \in R} r_{req}$ .

Ainsi la formulation du problème est présentée comme suit :

*Objectif* : $$\min((A*(X - \sum_{req \in R} r_{req})) + (B * \sum_{req \in R} \sum_{a \in A} C^{a} f_{req}^{a})$$

*Sous les contraintes :*

$$\sum_{req \in R} \sum_{a \in A} f_{req}^{a} \leq c^{a} \qquad \textbf{(1.3)}$$

*Pour chaque requête :* $(u \rightarrow v)$ : **(1.4)**

$$\sum_{(i,j) \in A} f_{req}^{(i,j)} = \sum_{(j,k) \in A} f_{req}^{(j,k)} \quad \text{Si i,j} \neq \text{u et i,j} \neq \text{v.}$$

$$\begin{cases} \sum_{(i,j) \in A} f_{req}^{(i,j)} = 0 & \text{Si j = u} \\ \sum_{(j,k) \in A} f_{req}^{(j,k)} = r_{req} * req\_size \end{cases}$$

$$\begin{cases} \sum_{(i,j) \in A} f_{req}^{(i,j)} = r_{req} * req\_size & \text{Si j = v} \\ \sum_{(j,k) \in A} f_{req}^{(j,k)} = 0 \end{cases}$$

$$\forall req \in R, \forall a \in A, f_{req}^{a} \geq 0$$
$$\forall req \in R, r_{req} \in \{0,1\}$$

Notre objectif étant la maximisation du nombre de demandes acceptées tout tenant compte du coût engendré, les contraintes se présentent comme suit :

- La contrainte n°**1.3** assure que le flot circulant sur un Arc « **a »** ne dépasse pas la capacité disponible sur cet arc.

- La contrainte n°**1.4** sert à assurer la conservation du flot pour les demandes de connexion acceptées.

*7.3.1.2- Algorithme formel proposé :*

L'algorithme formel proposé est le suivant :

**Algorithme2** Algorithme de routage avec prise en compte du futur

**Entrée :** $G = (V, E)$, *Duree_simulation*, $D = \{d_1, d_2, ..., d_m\}$ (ensemble des demandes de connexions triées dans un ordre croissant suivant leurs temps d'arrivée}, *nombre_demandes, Taille_fenêtre*
**Sortie :** Taux de réussite
**Variables locales :**
*connexions_finies_etape* : vecteur servant à stocker les connexion finies à une étape spécifiée.
*demandes_entrantes_etape* : vecteur servant à stocker les demandes de connexion générées à une étape spécifiée.
*Taux_réussite* : Pourcentage des demandes acceptées sur le réseau.
*pl_cplex :* Programme linéaire pour résoudre le problème.
*demandes_rejetées* : nombre de connexions rejetées tout au long de la simulation.
*taille_fenêtre :* taille fenêtre de vision (sert pour réduire la taille de fenêtre initiale)
*Etape_futur :* variable intermédiare ( compteur)

**1. Pour** chaque « etape » **allant de** 1 **à** *Duree_simulation* **faire**
**2.** Vider *connexions_finies_etape*
**3.** *connexions_finies_etape* ← *trouver_requêtes_sortantes*(*etape*)
**4.** **Si (***connexions_finies_etape* est non vide) **faire**
**5.** **Pour** chaque « *connexion* » de l'ensemble *connexions_finies_etape* **faire**
**6.** *Mettre_à_jour_capacités_et_coûts_des_liens_du_chemin_de*(*connexion*)
**7.** **Fin Pour**
**8.** **Fin Si**
**9.** *taille_fenêtre* ← *Taille_fenêtre*
**9.** *résolu*←**faux**
**10.** **Tant que**(*résolu* = **faux**) **faire**
**11.** **Si** (*demandes_générées_à_étape(étape)* **faire**
**12.** Vider *demandes_entrantes_etape*
**13.** *demandes_entrantes_etape* ← *demandes_entrantes_etape*
**14.** *créer_variables_cplex_des_demandes(étape, étape + taille_fenêtre)*
**15.** **Pour** *étape_future* **allant de** *étape* **à** *(étape + taille_fenêtre)* **faire**
**16.** *créer_flots_rlatifs_aux_demandes_générées_à(étape_future, pl_cplex)*
**17.** *Ajouter_contraintes_de_capacités(pl_cplex)*
**18.** **Fin Pour**

**19.** *créer_table_des_coûts_intermédiaires_des_liens()*
**20.** *Ajouter_objectif(pl_cplex)*
**21.** *résolu = resoudre(pl_cplex)*
**22.** **Si** (*résolu*)
**23.** *sauvegarder_résultats(requêtes_entrantes_étape)*
**24.** *taille_fenêtre = Taille_fenêtre*
**25.** **Sinon Si** (*taille_fenêtre* ≠ *0*) **faire**
**26.** *Réduire la taille de la fenêtre*
**27.** *résolu = faux*
**28.** **Sinon**
**29.** Résoudre le problème par le programme linéaire de maximisation du nombre de demandes acceptées tout en minimisant le coût et le nombre de demandes rejetées seront ajoutés à la variable *demandes_rejetées* et les requêtes acceptées seront sauvegardées avec leurs chemins.
**30.** *résolu = vrai*
**31.** *taille_fenêtre = Taille_fenêtre*
**32.** **Fin Si**
**33.** **Fin Si**
**34.** **Fin Tant que**
**35.** **Fin Pour**
**36.** $taux\ réussite \leftarrow \frac{(nombre\_demandes - demandes\_rejetées)}{nombres\_demandes} * 100$
**37.** **retourner**(*taux réussite*)

*7.3.1.3- Description détaillée de l'algorithme proposé :*

Après avoir extrait le scénario des demandes de connexions, un certain nombre de traitements doivent être appliqués pour mettre en œuvre l'algorithme proposé du routage avec prise en compte du futur :

Comme précédemment la première phase sera celle de :

- *Mettre à jour les capacités et les coûts sur les liens suite à la sortie des requêtes :*

Suite à la sortie d'une ou plusieurs requêtes, des unités de capacité vont être évidemment libérées sur les liens déjà empruntés par la (les) requête(s). Ainsi une opération de mise à jour de coût et de capacité est exigée au niveau de ces liens.

- *Créer les variables du modèle au niveau du programme linéaire :*

Pour résoudre le problème, il faut tout d'abord créer les différentes variables de décision. Les variables de décision dans notre exemple représentent le flot de chaque requête par lien et il faut créer toutes les variables relatives aux requêtes arrivant à l'étape courante et celles générées dans les étapes du futur (taille de la fenêtre).

- *La création du flot :*

La création du flot représente la mise en place da la contrainte n°2 de notre 1er programme linéaire : en d'autres termes la création revient à assurer la conservation des flots pour les nœuds transparents (intermédiaires) et à ajoute (ou retirer) du flot sur les liens incidentes à un nœud producteur (ou consommateur). La définition de la quantité du flot sera définie bien sûr lors de la production du flot. Un petit schéma descriptif pour voir mieux la problématique.

- *L'ajout des contraintes :*

Un ajout de contraintes est réalisé à chaque étape pour ne pas dépasser les capacités disponibles sur les liens à une étape donnée. En effet, la mise en place des contraintes doit tenir compte de la dynamique du trafic : l'ajout et la sortie des requêtes à chaque étape (s'il y a des ajouts  des sorties à cette étape bien sûr).

- *Créer table intermédiaire des coûts :*

Comme on a précisé dans le modèle, le coût relatif à chaque lien utilisé pour la résolution du problème est un coût approximatif calculé suivant le principe précédemment ainsi une table que j'ai appelé intermédiaire va contenir les coûts approximatif qu'on utilisera dans le modèle.

- *Ajouter l'objectif au programme linéaire :*

Spécifier comme objectif du programme linéaire : la minimisation du coût global de la solution trouvée.

- *Résoudre le programme linéaire :*

A ce niveau une solution (si elle existe) est renvoyée par le ***Solveur Cplex*** utilisé. A base de cette solution les demandes de connexion peuvent avoir leurs chemins spécifiques et ils seront ainsi mis sur le réseau. Sinon, nous réduisons un peu la taille de la fenêtre et on réexécute  toutes les étapes précédentes excepté da la première étape. Et si avec une taille de fenêtre égale à 1 on n'arrive pas à trouver une solution alors le problème sera la maximisation de nombre de requêtes acceptées (pour les requêtes générées à l'étape actuelle). A ce stade, un autre programme linéaire est crée avec de nouvelles contraintes et une solution est renvoyée après la résolution indiquant les requêtes qui ont été accepté.

## ***7.4- Expérimentations, résultats et interprétations :***

### *7.4.1- 1er scénario réalisé, Résultats et interprétations :*

Un scénario de trafic a été généré avec les paramètres suivants :

- *Nombres de requêtes* : 5000 requêtes.
- *Intervalle de séjour* : 800 étapes.
- $\beta$ : 200 étapes.
- *La fonction de coût utilisée est celle à* : croissance parabolique.

Le réseau utilisé pour la simulation est ***NSFNET*** avec « ***96 »*** unités de capacité sur chaque lien.

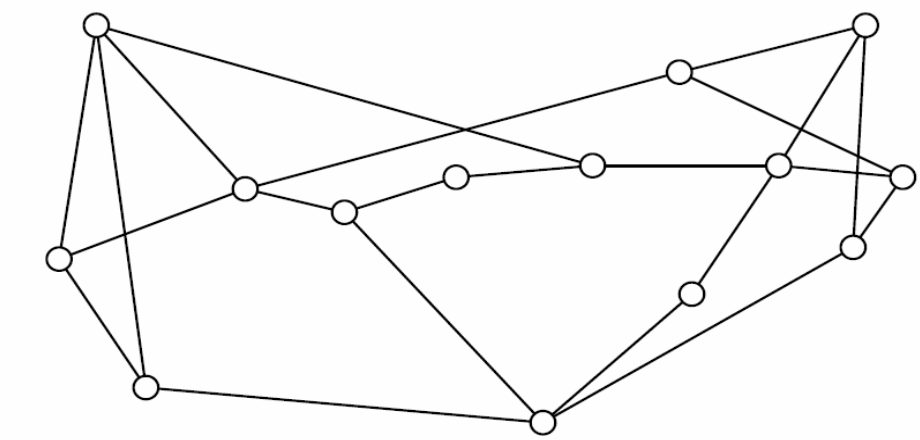


***Figure : réseau NSFNET***

- *Résultats recueillis pour le routage à myope pour le 1er scénario :*

Les résultats qu'on va exposer, représentent des valeurs moyennes d'une série de simulation et ce à cause des fluctuations remarquées au niveau les résultats observés (en répétant les même simulations). Ces fluctuations figurants à de l'exécution de l'algorithme du routage myope résultent de la diversité de solutions que peut trouver l'algorithme du plus court chemin « ***DIJKSTRA*** », et un choix aléatoire peut influencer les résultats trouvés à la fin de la simulation.

- *Taux de réussite* : ***87.15% ± 0.4%***

- *Résultats recueillis pour le routage avec prise en compte du futur pour le 1er scénario:*

En procédant aux simulations, j'ai obtenu comme résultats les valeurs suivantes :

| ***Liste des simulations réalisées :*** | ***Valeur moyenne:*** |
|---|---|
| ***Routage avec prise en compte de : 1 étape dans le futur.*** | ***89%*** ± 0,4% |
| ***Routage avec prise en compte de : 2 étapes dans le futur.*** | ***88,97%*** ± 0,2% |
| ***Routage avec prise en compte de : 4 étapes dans le futur.*** | ***88,97%*** ± 0,4% |
| ***Routage avec prise en compte de : 6 étapes dans le futur.*** | ***88,8%*** ± 0,1% |
| ***Routage avec prise en compte de : 10 étapes dans le futur.*** | ***89.19%*** ± 0,4% |
| ***Routage avec prise en compte de : 20 étapes dans le futur.*** | ***89%*** ± 0,4% |
| ***Routage avec prise en compte de : 50 étapes dans le futur.*** | ***89%*** ± 0,2% |

- *Interprétations des résultats relatives 1er scénario:*

A première vue, il apparaît que le routage avec prise en compte du futur assure toujours de meilleurs résultats que le routage myope : en effet on remarque une amélioration fluctuant entre 1% à 2.6% durant toutes les simulations réalisées. Ce qu'on peut remarquer aussi, c'est qu'on ne peut pas savoir à partir de quelle taille de fenêtre on peut atteindre la meilleure amélioration, ou laquelle des répétitions, pour une taille de fenêtre fixée, va nous assurer la meilleure amélioration. : Tout ça peut paraître logique du fait que :

- Les calculs des routes à une étape spécifiée sont basés sur la connaissance des ***k*** étapes suivantes mais on n'est pas certain qu'en tenant compte de la $k^{ème}+1$ étape dans le futur on préservera la même solution.

Et comme on a indiqué précédemment le routage avec prise en compte du futur n'a pu assuré que 2.6% comme amélioration maximale par rapport au routage myope. 2 interprétations sont possibles :

- Soit que nous avons atteint le seuil maximum d'amélioration pour la capacité du réseau.
  Ce qui n'est pas si facile à prouver : en effet pour être capable d'affirmer qu'on a utilisé à fond nos ressources on va présenter dans la section III l'approche de ***reconfiguration***.

- Soit que les écarts cumulés d'utilisation d'un coût approximé à limiter l'augmentation du taux de réussite en regardant plus dans le futur.

Pour assurer plus de précision sur les interprétations, j'ai essayé de regarder d'autres critères tel que la capacité totale restante sur le réseau et la capacité disponible restante sur le lien le plus congestionné à chacune des étapes de la durée de simulation :

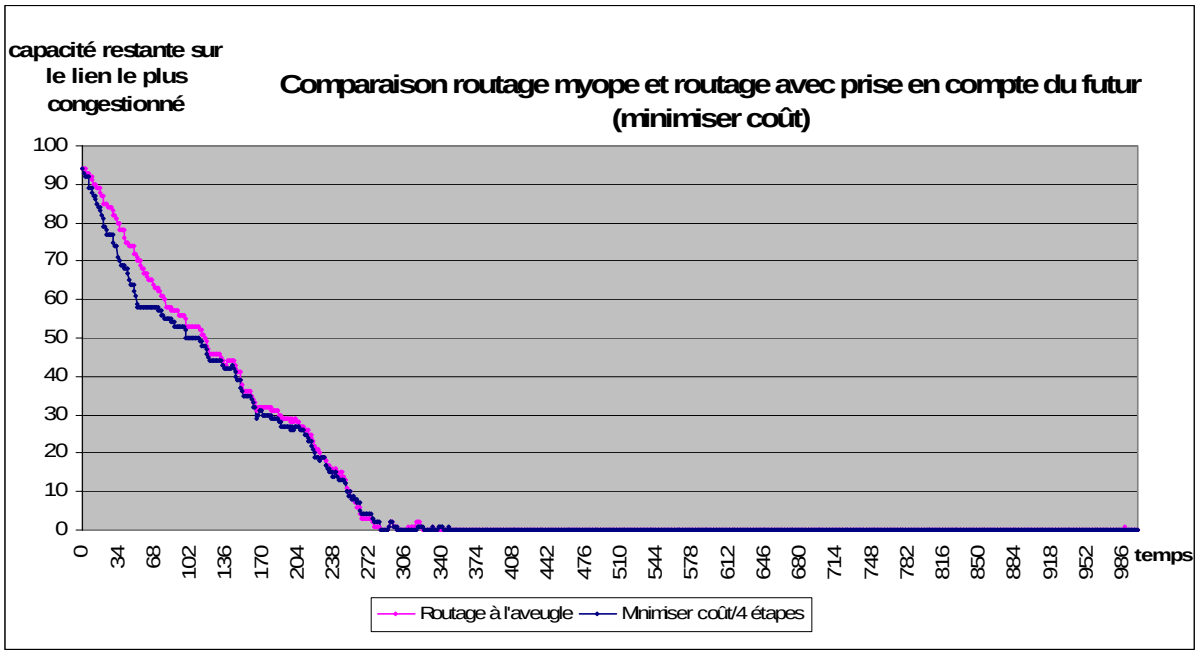



**Figure : 4**

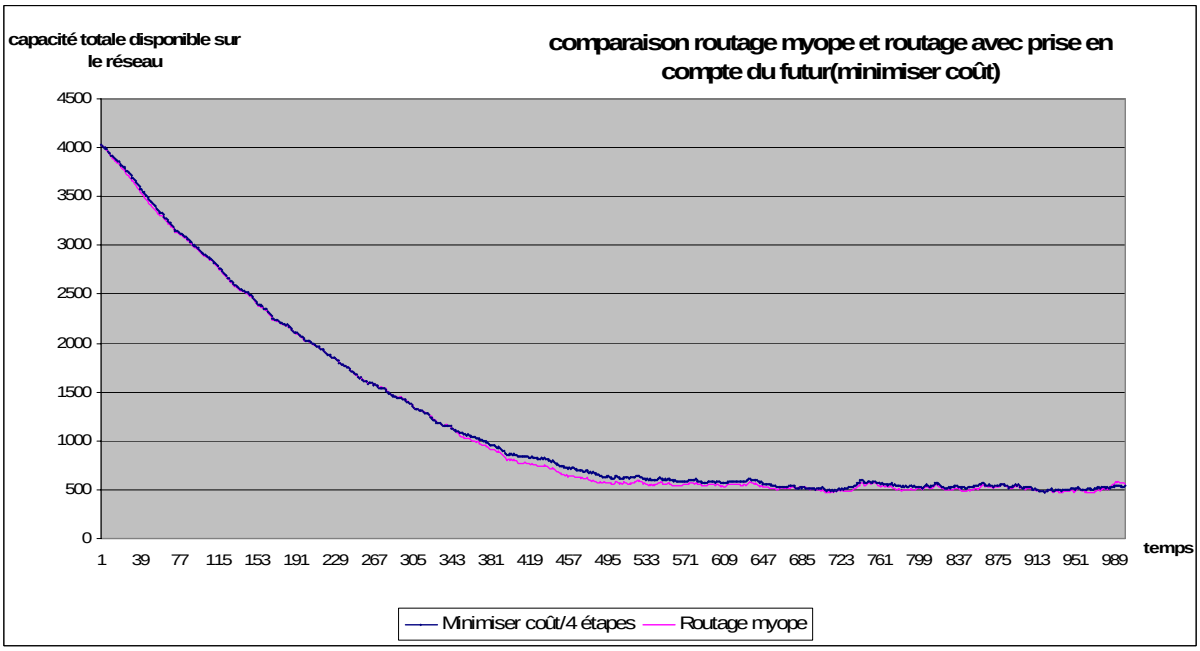



**Figure : 5**

A première vue, il est clair que les 2 algorithmes admettent le même comportement soit pour « la capacité globale restante sur le réseau» ou « la capacité disponible restante sur le lien le

plus congestionné ». Cette ressemblance très claire résulte du fait que les 2 algorithmes se basent sur l'objectif de minimisation du coût et qu'ils utilisent la même instance du trafic qui peut pour une topologie assez limité (n'admet pas trop de chemins disponibles pour une demande) influencer la décision des liens choisis.

La première figure, montre bien qu'on commence à utiliser à 100% un ensemble des lien du réseau à partir de l'étape n ° 350, et c'est à partir de cet étape qu'on commence réellement à rejeter des demandes (s'il ont bien sûr besoin d'utiliser ces liens).

Mais ce qui est vraiment intéressent à voir, c'est que là ou on commence à rejeter des demandes de connexions il reste toujours 500 unités de capacité sur le réseau en moyenne. Deux interprétations sont possibles :

- Soit que les demandes de connexions ont été mal servies (attribution non optimale de ressources) ce qui est très logique car pour le routage en myope on route les requêtes dés leurs arrivées et pour le routage avec prise en compte du futur notre vision n'est pas si étendue pour englober tout le futur.

- Soit que la topologie du réseau nous a amené à de tels résultats, c'est qui peut être aussi logique de fait que sur le réseau *NSFNET* on remarque une connectivité assez particulière décrite comme suit :

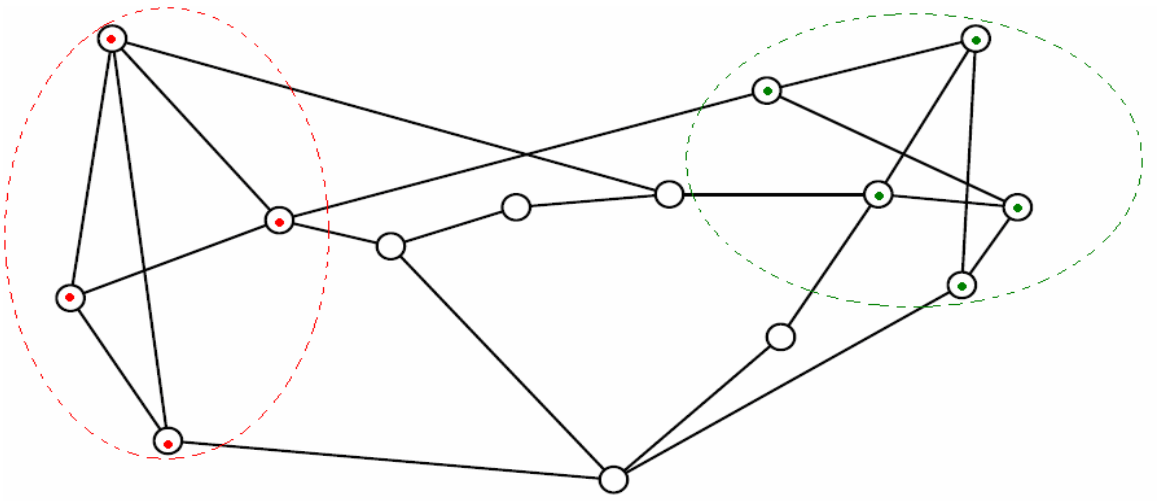


**Figure : 6**

Pour les demandes de communications partant des nœuds rouges aux noeuds verts et vice vers ça, les routes choisies sont assez longues (plus que 2 liens) et passent toujours par les mêmes liens du milieu, ce qui va être considéré à un certain moment comme un goulot d'étranglement pour le réseau. Mais pour les cotés à droite et à gauche, on peut effectivement conserver de la capacité disponible ce qui peut être ainsi la cause des résultats observées précédemment au niveau de la figure reflétant la capacité globale disponible sur le réseau tout au long de la simulation.

Pour cerner plus l'espace des interprétations, j'ai refait les mêmes expériences sur un autre type de réseau appelé ***cost239*** ayant d'autres caractéristiques que le ***NSFNET*** :

*7.4.2- 2eme scénario réalisé, Résultats et interprétations :*

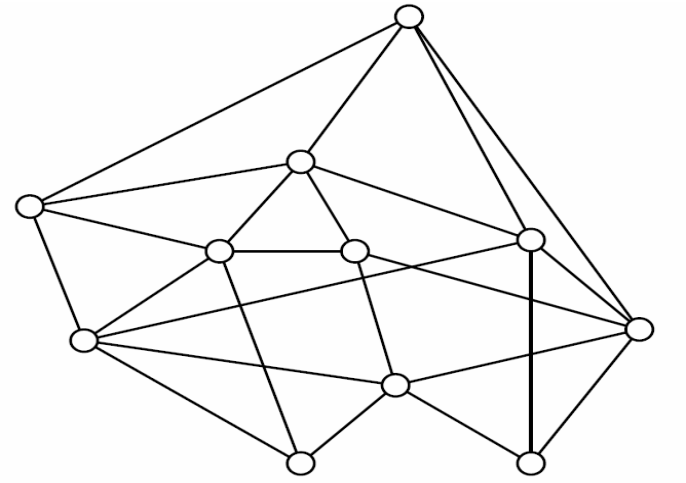


***Figure : réseau Cost239***

Il est clair que ce réseau a un diamètre 3 ($\forall x, y \in V * V$ , on peut trouver du sommet ***x*** au sommet ***y*** un chemin de longueur inférieure ou égale à 3).

Pour le même scénario de trafic présenté précédemment :

- *Nombres de requêtes* : 5000 requêtes.
- *Intervalle de séjour* : 800 étapes.
- $\beta$ : 200 étapes.

Le réseau ***cost239*** avec « ***96 »*** unités de capacité sur chaque lien, a pu assurer le routage de toutes les demandes de connexions du scénario, d'où nous étions obligés soit de :

- minimiser la capacité disponible sur chacun des liens du réseau.
- Soit d'augmenter le trafic généré.

La première proposition a été retenue, et on a réduit la capacité des liens à ***64*** unités mais on n'a pas eu assez de blocage, d'où on a réduit de plus à ***48*** unités les résultats suivants ont été visualisés :

(Il est à remarquer que la formulation de notre problème peut nous servir pour résoudre d'autres problèmes intéressants, tel que le problème de « ***dimensionnement des réseaux*** ». En effet, on a remarqué pour le réseau « ***cost239*** » avec des liens de ***96 unités*** de capacité

notre réseau était surdimensionné en terme de ressources disponibles. D'où on a recourt à minimiser le nombre de ressources utilisées).

- *Résultats recueillis pour le routage à myope pour le réseau cost239 :*

  - *Taux de réussite* : ***73.8%*** ± ***0.3%***

- *Résultats recueillis pour le routage avec prise en compte du futur pour le réseau cost239 :*

En procédant aux simulations, j'ai obtenu comme résultats les valeurs suivantes :

| ***Liste des simulations réalisées :*** | ***Valeur moyenne:*** |
|---|---|
| ***Routage avec prise en compte de : 1 étape dans le futur.*** | ***76%*** ± 0,1% |
| ***Routage avec prise en compte de : 2 étapes dans le futur.*** | ***75,9%*** ± 0,3% |
| ***Routage avec prise en compte de : 4 étapes dans le futur.*** | ***76%*** ± 0,3% |
| ***Routage avec prise en compte de : 6 étapes dans le futur.*** | ***76.4%*** ± 0,3% |
| ***Routage avec prise en compte de : 10 étapes dans le futur.*** | ***76.4%*** ± 0,3% |
| ***Routage avec prise en compte de : 20 étapes dans le futur.*** | ***76.3%*** ± 0,3% |
| ***Routage avec prise en compte de : 50 étapes dans le futur.*** | ***76.15%*** ± 0,8% |

- *Interprétations des résultats relatives au réseau cost239:*

A première vue, et comme il a été remarqué sur la topologie précédente le routage avec prise en compte du futur assure toujours de meilleurs résultats que le routage myope : en effet on remarque une amélioration fluctuant entre 1.5% à 2.8% durant toutes les simulations réalisées. De même, ce qu'on peut remarquer aussi, c'est qu'on ne peut pas savoir à partir de quelle taille de fenêtre on peut atteindre la meilleure amélioration, ou laquelle des répétitions, pour une taille de fenêtre fixée, va nous assurer la meilleure amélioration.

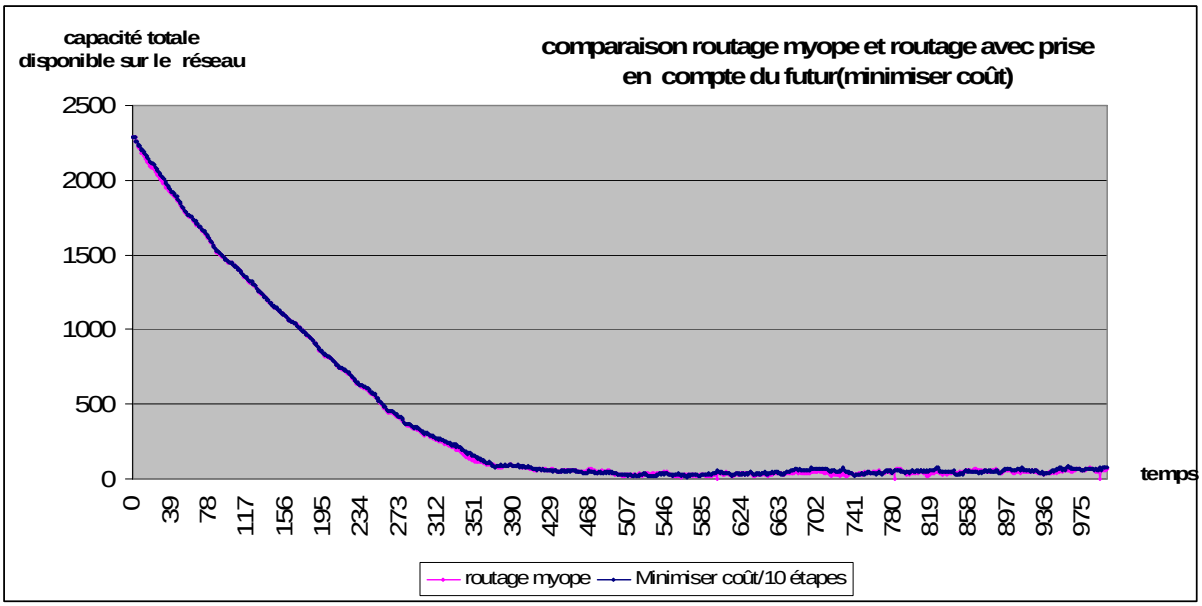



**Figure : 6**

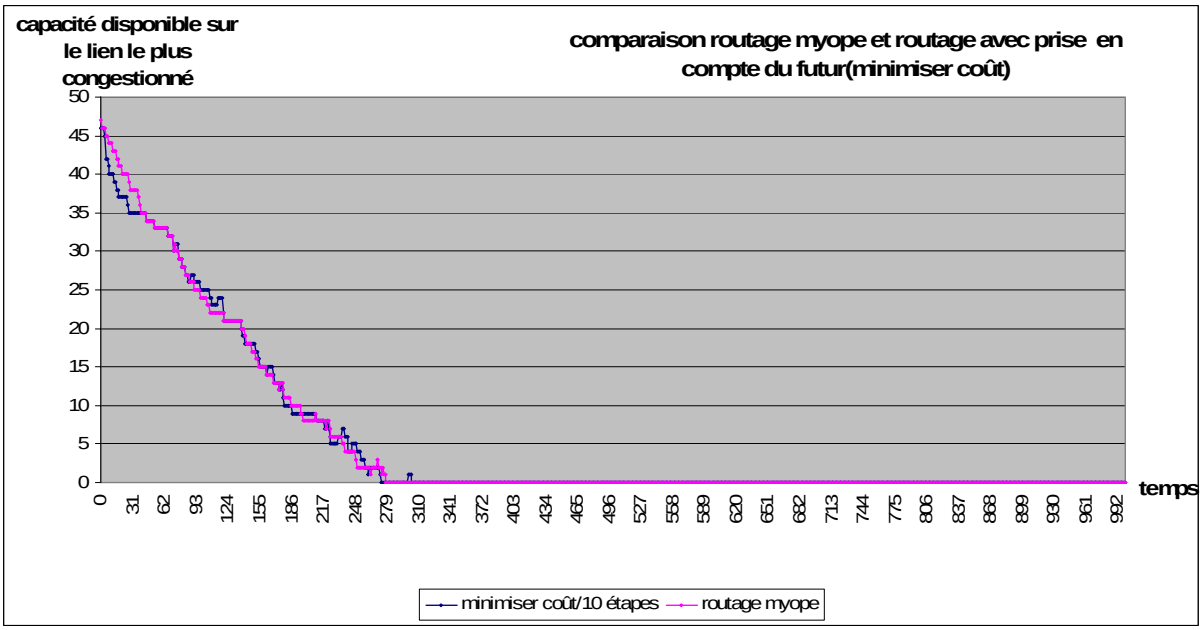



**Figure : 7**

Globalement le comportement des algorithmes sur les 2 topologies : NSFNET et cost239 est le même. La seule différence assez claire se manifeste au niveau de capacité globale restante : en effet pour le réseau NSFNET là où on commence à rejeter des demandes de connexions il reste à peu prés 12.5% de la capacité globale disponible tandis que pour le réseau cost239 là où on commence à rejeter des demandes de connexion il reste en moyenne sur le réseau 1% de la capacité globale offerte initialement. Tout ça revient en fait à la différence de topologies

entre les 2 réseaux. Mais ça nous donne aucune garantie sur la capacité des réseau à accepter plus de demandes de connexions ; pour être sûr qu'on peut améliorer ou non le taux d'acceptation on a eu recourt à la reconfiguration qu'on va présenter dans la partie qui suit.

## ***7.4- Reconfiguration:***

Dans un réseau optique WDM reconfigurable, il est possible de déplacer une connexion (canal de communication optique entre 2 noeuds du réseau, généralement à plus de 2.5Gbit/s) vers une nouvelle route. L'évolution du trafic (ajout/suppression de connexions) au cours du temps entraîne généralement une mauvaise utilisation des ressources du réseau. Une reconfiguration du réseau consiste alors à déplacer des connexions pour retrouver une utilisation optimale (ou proche de l'optimale) des ressources. Ainsi, le problème de reconfiguration consiste à définir de quelle façon faire évoluer la configuration de la couche optique en fonction des évolutions du trafic au cours du temps.

L'approche de reconfiguration certes, permet une meilleure attribution, mais il est à savoir que le déplacement d'une connexion comporte des risques tel que l'arrêt du trafic, la perte d'informations, ... de ce fait, les travaux qui ont été proposés, pour assurer la meilleure mise en place d'une telle approche, essaye généralement de trouver un compromis entre le nombre de changements de connexions effectués et le taux d'amélioration de la solution de routage courante (une reconfiguration totale n'est pas conseillé vue le temps d'interruption résultant).

A notre niveau, on n'a pas à s'intéresser à tout ces problèmes et on ne va pas présenter les travaux proposés en littérature ; on va juste utiliser l'approche de reconfiguration totale déclenchée suite à des situations bien précises à fin de savoir quel taux d'acceptation peut-on atteindre avec le scénario de trafic et une des topologies de réseau présentés précédemment.

### *7.5.1- La reconfiguration totale :*

Etant donné un réseau $G = (V, E)$ et une instance de trafic $D = \{d_1, d_2, ..., d_m\}$ (avec $d_i$ : une connexion établie entre deux nœuds du réseau) actuellement sur ce réseau. Pour assurer une meilleure attribution des ressources (par exemple : minimiser le nombre de ressources utilisés ou libérer plus de la capacité sur les liens) une reconfiguration peut être déclenché. Ainsi, on peut lancer un algorithme de reconfiguration suite au dépassement d'un seuil de congestion, déjà définie, au niveau d'un des liens des réseaux ou après chaque $\boldsymbol{k}$ (entier quelconque) étapes (périodiquement). Pour ce qui nous concerne, suite à un rejet de demande de connexion notre algorithme de reconfiguration se déclenchera pour assurer, bi en sûr si c'est possible, le routage de cette demande. Pour le cas du routage normal des demandes de connexion un des algorithmes présentés précédemment (routage avec prise en compte du futur ou routage myope) vont être utilisés. D'où, un algorithme formelle de *routage avec reconfiguration* peut être présenté comme suit :

**Algorithme3** Algorithme de routage avec reconfiguration

**Entrée :** $G = (V, E)$, *Duree_simulation*, *D* {ensemble des demandes de connexions triées dans

L'ordre croissant de leurs temps d'arrivé}, *nombre_demandes.*

**Sortie :** Taux de réussite

**Variables locales :**

*Connexions_sur_le_reseau* : vecteur servant à stocker les connexions établies sur le réseau l'étape courante.

*Nombre_de_connexions_sur_réseau* : nombre de connexions sur le réseau à l'étape courante.

*Nombre_de_connexions_acceptées* : le nombre de connexions acceptés après reconfiguration.

*Nombre_requêtes_rejetées* : nombre des requêtes rejetées durant la simulation

**1. Pour** chaque « etape » de la *Duree_simulation* **faire**
**2.** **Pour** chaque « *requête* » générée à cette étape **faire**
**3.** **Si** *(routage possible de cette requête)* **alors** *(router et faire les mises à jour nécessaires)*
**4.** **Sinon**
**5.** *Sauvegarder l'attribution courante des chemins*
**6.** *Ajouter « requête » à « connexions_sur_le réseau »*
**7.** *Nombre_connexions_sur_réseau ← Nombre_connexions_sur_réseau + 1*
**8.** *Nombre_de_connexions_acceptées = reconfiguration(connexions_sur_réseau, G)*
**9.** **Si (***Nombre_de_connexions_acceptées ≠ Nombre_de_connexions_sur_réseau)* **Alors**
**10.** *Nombre_requêtes_rejetées ← Nombre_requêtes_rejetées + 1*
**11.** *Nombre_connexions_sur_réseau ← Nombre_connexions_sur_réseau - 1*
**12.** *Préserver la solution initiale (avant le lancement de reconfiguration)*
**13.** **Sinon**
**14.** *Extraire la liste des chemins résultante de la reconfiguration*
**15.** *Faire les mises à jour nécessaires*
**16.** **Fin Si**
**17.** **Fin Si**
**18.** **Fin Pour**
**19. Fin Pour**
**20.** *taux réussite* $\leftarrow \frac{(nombre\_demandes - requêtes\_rejetées)}{nombres\_demandes} * 100$
**21. retourner**(*taux réussite*)

*7.5.2- Description détaillée de l'algorithme proposé :*

Etant donné que nous avons déjà traité en détail avant l'algorithme de routage, on s'intéressera dans cette partie à expliquer la méthodologie utilisée pour mettre an place l'approche de reconfiguration totale :

- *Sauvegarder l'attribution courante des chemins :*

Une reconfiguration ne garantit pas toujours l'amélioration d'une attribution de chemins courante, de ce fait il faut toujours, avant de la déclencher, assurer la sauvegarde de l'attribution courante.

- *Déclencher l'algorithme de reconfiguration :*

On entend dire par la reconfiguration ici : « la maximisation de nombres de connexions acceptées sur un réseau existant avec des ressources bien définies ». En effet, dans notre cas, le travail consiste à :

- Retirer les connexions existantes déjà sur le réseau : donc remettre les capacités disponible sur l'ensemble des liens et leurs coûts à leurs valeurs initiales.
- Ajouter à ces connexions la nouvelle demande de connexion.
- Essayer de router cette instance sur le réseau (déjà vide).

Pour le dernier point, un programme linéaire a été formulé à fin de maximiser le nombre de connexions acceptées sur le réseau et le voici (on n'a changé aucune notation déjà définie):

Objectif : $\max \sum_{req \in R} r_{req}$

Sous les contraintes :

$$\sum_{req \in R} \sum_{a \in A} f_{req}^{a} \leq c^{a} \qquad \textbf{(2.3)}$$

*Pour chaque requête :* $(u \rightarrow v)$ : **(2.4)**

$$\sum_{(i,j) \in A} f_{req}^{(i,j)} = \sum_{(j,k) \in A} f_{req}^{(j,k)} \quad \text{Si i,j} \neq \text{u et i,j} \neq \text{v.}$$

$$\begin{cases} \sum_{(i,j) \in A} f_{req}^{(i,j)} = 0 & \text{Si j = u} \\ \sum_{(j,k) \in A} f_{req}^{(j,k)} = r_{req} * req\_size \end{cases}$$

$$\begin{cases} \sum_{(i,j) \in A} f_{req}^{(i,j)} = r_{req} * req\_size & \text{Si j = v} \\ \sum_{(j,k) \in A} f_{req}^{(j,k)} = 0 \end{cases}$$

$$\forall req \in R, \forall a \in A, f_{req}^{a} \geq 0$$
$$\forall req \in R, r_{req} \in \{0,1\}$$

Un tel programme linéaire, va retourner nécessairement les valeurs des variables de décisions $r_{req}$ et ainsi on saura le nombre de connexions acceptées et pour les chemins leur identification sera faite à travers les valeurs attribuées aux *«variables de flot relatives à chaque requête sur chaque lien ».*

Et si le nombre de connexions acceptées est égal au nombre de connexions formulant l'instance introduite alors on est arrivé à assurer une route valide à la demande de connexion rejetée par notre algorithme de routage, sinon le réseau n'a plus de capacité disponible pouvant servir le routage de cette demande de connexion et elle sera forcément rejeté.

Et comme il a été indiqué dans l'algorithme, si la solution trouvé arrive à assurer une route à la demande de connexion déjà rejetée alors :

- On applique la solution trouvée au réseau : on attribue les chemins aux demandes de connexions et on effectue les mises à jours nécessaires de coût et de capacité sur les liens.

Sinon :

- On applique de nouveau la solution déjà sauvegardée avant le déclenchement de l'algorithme de reconfiguration.

*7.5.3-Suite des expérimentations, résultats et interprétations :*

- l'algorithme de routage utilisé est celui du routage ***Myope***
- Le scénario utilisé est le scénario n° 1 sur le réseau ***NSFNET***

Avec telles conditions, on a pu assurer un taux d'acceptation égale à ***93%.*** Ce qui affirme l'idée qu'on n'a pas atteint la meilleure solution avec l'algorithme du « *routage avec prise en compte du futur* » formulé comme précédemment. A ce niveau, une question pertinente peut être posé :

- *Question : Peut on faire mieux ?*

## *7.6- Routage avec prise en compte du futur, 2eme Modèle proposé :*

*7.6.1- Deuxième modèle proposé : minimisation de la congestion :*

Pour essayer de répondre à la question posée, il faut améliorer la prise en compte du futur. Nous avons développé un deuxième mode, plus précis, permettant la minimisation de la congestion.
L'objectif de ce modèle est de préserver de la capacité sur tous les liens des réseaux tant que c'est faisable. En effet, si on dispose d'une unité de capacité sur tous les liens du réseau, alors on peut trouver une route pour toute nouvelle demande de connexion.

*7.6.2- Description du 2emé modèle proposé :*

Pour la formulation et les notations du problème on ajoute quelques paramètres :

- usedCapacity$^a$(t) : la capacité utilisé sur une arête « ***a*** » à une étape donnée. $\forall a \in A$.
- $capacityOnEdge^a$ : la capacité d'une arête « ***a*** ». $\forall a \in A$.
- $\lambda$ : taux d'utilisation de lien.

Objectif : $\min \lambda$

Sous les contraintes :

$$usedCpacity^a(t) + (\sum_{t=T1}^{t} \sum_{req \in R_E(t)} f_{req}^a - \sum_{i=T_1}^{t} \sum_{req \in R_S(t)} f_{req}^a) \leq \lambda * capacityOnEdge^a \qquad T_1 < t \leq T_1 + k \quad \textbf{(2.1)}$$

$$\sum_{(u,v) \in A} \sum_{req \in R_E(t)} f_{req}^{(u,v)} + pr(v, req) = \sum_{(v,u) \in A} \sum_{req \in R_E(t)} f_{req}^{(v,u)} + co(v, req) \qquad T_1 < t \leq T_1 + k \quad \textbf{(2.2)}$$

$$\forall req \in R, \forall a \in A, f_{resq}^a \geq 0$$
$$\lambda \in \{0,1\}$$

Comme il a été décrit dans le modèle de minimisation du coût, il est à noter que dès que des arcs seront très congestionnés, des requêtes seront rejetées. Pour essayer de les router, on essayera de relâcher les contraintes de prise en compte du futur en réduisant la taille de la fenêtre, éventuellement à 1. Mais dans le modèle suivant proposée on ne se soucie plus de la contrainte de minimisation de coût : en effet on ne va s'intéresser qu'à la maximisation du nombre de demandes de connexion acceptées.

Le modèle proposé est le suivant …….

*7.6.3- Suite des expérimentations, résultats et interprétations :*

Premièrement on était impressionné par le temps énorme mis pour résoudre un tel programme linéaire : en effet due au temps énorme mis on était obligé :

- A se contenter de solutions qui sont loin de 10% de la solution optimale.
- De ne pas aller trop dans le futur suite au temps énorme mis pour la résolution (au maximum 5 étapes)

Les simulations ont été réalisées sur le même modèle décrit précédemment avec le réseau *NSFNET* : Les résultats recueillis n'étaient pas meilleurs que les précédents, on a eu un taux de réussite qui varie entre 78% et 79% et ça nous a parus logique car l'algorithme utilisé autorise pour une demande de connexion quelconque de choisir des chemins si longs tant que ça n'augmente pas le taux d'utilisation des liens ainsi on s'est retrouvé avec des demandes

de connexions avec des durées assez grandes éparpillées sur une grande portion des liens du réseau ce qui complique plus les choses lors du blocage (où on commence à rejeter des requêtes) du fait que les liens vont mettre plus de temps pour être libérer.

Pour assurer plus de précision sur les interprétations, j'ai essayé de regarder, comme j'ai fait pour le modèle de minimisation du coût, d'autres critères tel que la capacité totale restante sur le réseau et la capacité disponible restante sur le lien le plus congestionné à chacune des étapes de la durée de simulation :

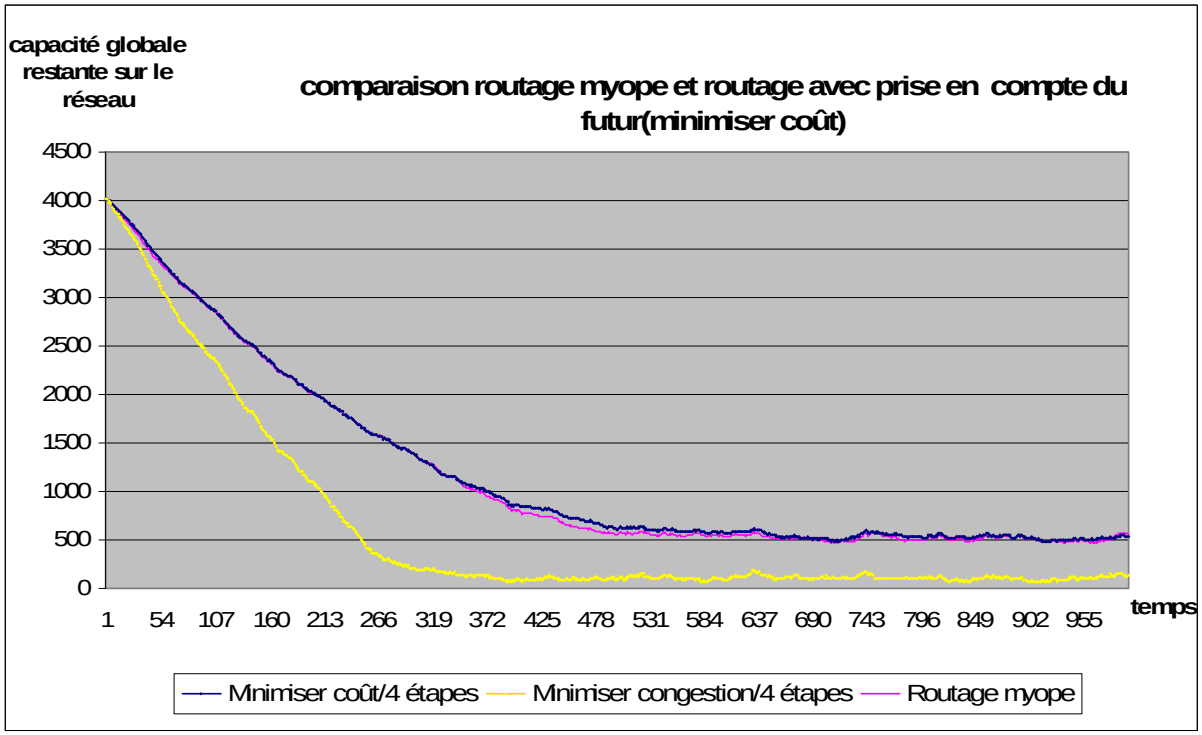



**Figure : 7**

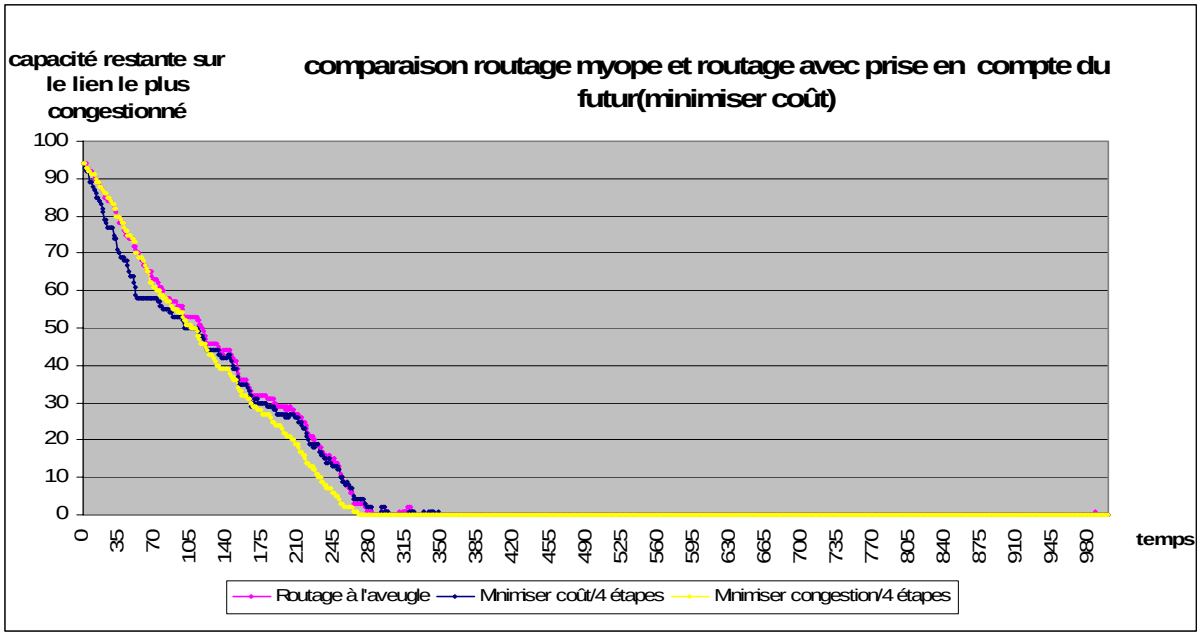



**Figure : 8**

La figure n° 7 explique clairement pourquoi on a eu tant de rejet de demandes de connexions pour le modèle de minimisation de congestion. En effet, en utilisant ce dernier le réseau conserve moins de capacité globale qu'en utilisant les 2 autres algorithmes et cela résulte en fait de la nature du modèle, qui comme on a déjà expliqué autorise le choix de chemins assez long tant que ça n'augmente pas le taux courant d'utilisation des liens.

Une autre chose qui a peut être causer de tel résultats est qu'on n'a pas pensé à assurer plus de capacité disponible à la fin de notre champ de vision dans le futur, en quelque sorte on essaye de minimiser le dernier taux d'utilisation de liens de notre champ de vision dans le futur au lieu de minimiser le taux global d'utilisation de liens.

### *7.7- Routage avec prise en compte du futur, 3ème Modèle proposé :*

Il est à noter que le seul changement remarqué par rapport au modèle précédent est qu'on a un taux d'utilisation au niveau de chaque étape dans le futur et on va essayer de minimiser le dernier :

- $\lambda^t$ : taux d'utilisation de lien.

*Objectif :* $\min \lambda^{T1+k}$

*Sous les contraintes :*

$$usedCpacity^a(t) + \left(\sum_{t=T1}^{t} \sum_{req \in R_E(t)} f_{req}^a - \sum_{i=T_1}^{t} \sum_{req \in R_S(t)} f_{req}^a\right) \le \lambda^t * capacityOnEdge^a \qquad T_1 < t \le T_1 + k \quad \textbf{(3.1)}$$

$$\sum_{(u,v)\in A}\sum_{req\in R_E(t)} f_{req}^{(u,v)} + pr(v,req) = \sum_{(v,u)\in A}\sum_{req\in R_E(t)} f_{req}^{(v,u)} + co(v,req) \qquad T_1 < t \le T_1 + k \quad \textbf{(3.2)}$$

$$\forall req \in R, \forall a \in A,\ f_{resq}^{a} \ge 0$$
$$\lambda \in \{0,1\}$$

Mais tant qu'on ne peut plus aller trop dans le futur suite à la complexité du problème, on n'aura pas trop de variation durant notre fenêtre de prise en compte du futur (rien que l'ajout de requêtes sur le réseau ou la sortie de quelques requêtes déjà routé : un nombre assez petit) d'où on va se ramener presque au problème précédent et on n'aura pas une amélioration assez grande.

# Chapitre 8

# CONCLUSIONS ET PERSPECTIVES

# Bibliographie


[1] I. Chlamtac, A. Ganz, and G. Karmi, "Lightpath communications: An approach to high bandwidth optical WAN's". IEEE Transactions on Communications, 40(7) : 1171-1182 July 1992

[2] R. K. Pankaj. Architectures for Linear Lightwave Networks. PhD thesis, Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 1992.

[3] H. Zang, J. P. Jue, B.Mukherjee. A Review of Routing and Wavelength Assignment Approaches for Wavelength-Routed Optical Networks. OPTICAL NETWORKS MAGAZINE, January 2000.

[4] K. Chan and T. P. Yum, "Analysis of least Congested Path Routing in WDM Lightwave Networks," Proc., IEEE INFOCOM '94, Toronto, Canada, vol. 2, pp. 962-969, April 1994.

[5] H. Harai, M. Murata, and H.Miyahara, "Performance of Alternate Routing Methods in All-Optical Switching Networks," Proc., IEEE INFOCOM '97, Kobe, Japan, vol. 2, pp. 516-524, April 1997.

[6] L. Li and A. K. Somani, "Dynamic Wavelength Routing Using Congestion and Neighborhood Information," IEEE/ACM Transactions on Networking, to appear, 1999.

[7] S. Ramamurthy and B. Mukherjee, "Fixed-Alternate Routing and Wavelength Conversion in Wavelength-Routed Optical Networks," Proc., IEEE GLOBECOM '98, vol. 4, app. 2295-2302, Nov 1998.

[8] Théorie des graphes : http://gilco.inpg.fr/~rapine/Graphe/

[9] X. Zhang and C. Qiao, « Wavelength Assignment for Dynamic Traffic in Multi-fiber WDM Networks," Proc., 7th International Conference on Computer Communications and Networks, Lafayette, LA, LA, pp. 479-485, Oct. 1998

[10] S. Xu, L. Li and S. Wang, « Dynamic Routing and Assignment of Wavelength algorithms in Multi-fiber WDM Networks," IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 18, N°. 10, OCTOBER 2000.

[11] Optimisation combinatoire http://fr.wikipedia.org/wiki/Optimisation_combinatoire

[12] Classes de complexité. http://www.bibmath.net/dico/index.php3?action=affiche&quoi=./c/complexite.html