On the use of network coding and multicast for enhancing performance in wired networks

Yuhui Wang

THESE DE DOCTORAT CONJOINT TELECOM SUDPARIS et
L'UNIVERSITE PIERRE ET MARIE CURIE

Spécialité :

INFORMATIQUE et TELECOMMUNICATIONS

Ecole doctorale : Informatique, Télécommunications et Electronique de Paris

Présentée par

Yuhui Wang

Pour obtenir le grade de
DOCTEUR DE TELECOM SUDPARIS

Sur l'utilisation du codage réseau et du multicast pour améliorer la performance dans les réseaux filaires

Soutenue le 17 mai, 2013

devant le jury composé de :

Rapporteur : Fabio Martignon	Professeur, Université Paris-Sud
Rapporteur : Dritan Nace	Professeur, Université de Technologies de Compiègne
Examinateur : Prosper Chemouil	Directeur de Recherche, Orange Labs
Examinateur : Philippe Chretienne	Professeur, Université Pierre et Marie Curie
Examinateur : Muriel Médard	Professeur, Massachusetts Institute of Technology
Encadrant de thèse : Eric Gourdin	Ingénieur de Recherche, Orange Labs
Directeur de thèse : Eitan Altman	Directeur de Recherche, INRIA Sophia Antipolis
Directeur de thèse : Tijani Chahed	Professeur, Télécom SudParis

Thèse n° 2013TELE0010

DOCTORATE JOINTLY DELIVERED BY TELECOM SUDPARIS and PIERRE ET MARIE CURIE UNIVERSITY

Speciality :

INFORMATIQUE et TELECOMMUNICATIONS

Ecole doctorale : Informatique, Télécommunications et Electronique de Paris

Presented by

Yuhui Wang

For obtaining the
DOCTOR OF PHILOSOPHY DEGREE OF TELECOM SUDPARIS

On the Use of Network Coding and Multicast for Enhancing Performance in Wired Networks

Defended on 17^{th} May, 2013

Defense Committee :

Reviewer : Fabio Martignon	Professor, Université Paris-Sud
Reviewer : Dritan Nace	Professor, Université de Technologies de Compiègne
Examiner : Prosper Chemouil	Research Director, Orange Labs
Examiner : Philippe Chretienne	Professor, Université Pierre et Marie Curie
Examiner : Muriel Médard	Professor, Massachusetts Institute of Technology
Supervisor : Eric Gourdin	Research Engineer, Orange Labs
Supervisor : Eitan Altman	Research Director, INRIA Sophia Antipolis
Supervisor : Tijani Chahed	Professor, Télécom SudParis

Thesis n° 2013TELE0010

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

Yuhui Wang

Acknowledgements

To be honest, this research project would not have been possible without the support of many people.

First of all, I wish to express my most sincere gratitude and appreciation to my supervisor, Eric Gourdin for his invaluable guidance, endless patience, and constant encouragement through the entire journey in Orange Labs. I started working with him from the mid of my first year while I was in the most difficult time. His assistance helps me finding my self-confidence back, and restoring enthusiasm on the project.

Deepest gratitude are also due to the other Co-supervisors Pr. Eitan Altman, Pr. Tijani Chahed, and Nidhi Hedge. It was a great pleasure to have had a chance to work with Eitan, I would like to thank him for his thoughtful advice and attention. No words can express my appreciation to Tijani, without his knowledge and assistance, this study would not have been successful. Although time for the cooperation with Nidhi is short, but it is still memorable. Without her, I would not have had a chance to come to Paris.

My grateful thanks are also extended to my project manager Prosper Chemouil for his constant support and his good sense of humor.

Special thanks to Pr. Muriel Médard, without her invitation, I would not have had a chance to visit MIT and her excellent research lab. The short visit was an unforgettable and invaluable experience. Her insight and advice have kept me away from the wrong direction.

I would like to thank Adam Ouorou for his unlimited kindness. I appreciate that He has given me invaluable lessons of Operational Research in Telecommunication.

I am also indebted to all my France Telecom colleagues, especially my current team manager Nabil Benameur, former team manager Sara Oueslati, and all the team members, Pierre Bauguion, Amal Benhamiche, Alexandre Blogowski, Yannick Carlinet, Jean-Baptiste Dumont, Christine Gabet, Florence G. Benezit, Massimo Gallo, Hassan Hijazi, Raluca-Maria Indre, Bruno Kauffmann, Sharique Ali Khan, Thibaut Lefebvre, Fabien Mathieu, Jean-Robin Medori, Luca Muscariello, Philippe Olivier, Nancy Perrot, Alain Simonian, Jean-Mathieu Segura, and Christian Tanguy, for their invaluable and insightful comments during the field work and the write-up of the thesis.

I would like to express my very great appreciation to Pr. Fabio Martignon, Pr. Dritan Nace, and Pr. Philippe Chretienne for reviewing my thesis and being on the defense committee. Their willingnesses to give their time so generously have been very much appreciated.

I would like to thank Shule Song for her love and encouragement. The stability she helped to provide at home significantly eased the task of persevering through my studies. Her seemingly bottomless well of energy and joy have always refreshed and sustained me.

Last but not the least, I wishes to express my deepest love and heartfelt gratitude to my mother Ningzhi Yi, and Father Yanzhong Wang, for their unwavering support and unconditional love, through the duration of my studies.

Yuhui Wang
Orange Labs R&D, Issy-les-Moulineaux
February 2012

Résumé

Contexte

D'après le livre blanc de Cisco [3], le trafic IP mondial devrait tripler d'ici à 2016; certains trafics en particuliers, tels que la Vidéo à la Demande (VoD), la télévision sur IP, et les jeux en ligne, devrait connaitre une croissance spectaculaire dans les cinq prochaines années. Par conséquent, les opérateurs de réseaux devront faire d'importants investissements afin d'augmenter la capacité des réseaux et ainsi satisfaire les futures demandes. Pour attńuer l'impact de ces futurs investissements, il devient donc de plus en plus impératif de mieux maitriser l'écoulement du trafic dans les réseaux afin d'utiliser au mieux les ressources disponibles.

L'une des plus importantes caractéristiques des services tels que la télévision IP ou les jeux en ligne, réside dans le fait que les données doivent toujours être accessible simultanément par un grand nombre d'utilisateurs. Lorsque ces applications utilisent un des protocoles les plus répandues dans les réseaux de télécommunication actuels, tel que *unicast* ou *broadcast*, de très nombreuses copies des données initiales sont générées et doivent être écoulées dans le réseau, ce qui conduit à une très mauvaise gestion des ressources du réseau. Avec un protocole de routage dits unicast, l'échange de données se fait au moyen d'une connection établie entre une source et une destination à la fois. Malgré cette limitation intrinsèque, les protocoles unicast sont encore massivement utilisés dans le réseaux actuels.

Si plusieurs terminaux requièrent simultannément les mêmes données, et ce à partir d'une source unique, plusieurs sessions unicasts doivent être établies en parallèle, et les mêmes données peuvent donc être amenées à transiter dans le même lien. Au contraire, avec un protocole de type broadcast, un nœud de transit peut dupliquer les données reçues par une interface et les propager vers chacune de ses interfaces de sortie, ou dans une plage de communication dans un réseau sans fil (on parle parfois de "technique d'inondation"). Bien que ce

protocole garantisse bien que les données émises par la source seront finalement reçues par chaque terminal demandeur, l'information transmise risque d'inonder le réseau et d'épuiser les ressources réseaux. Afin de réduire les flux de données pour des applications réseaux qui entre une source et plusieurs destinations, le mode de transmission appelé multicast a été introduit à la fin des années 80. Pour mieux gérer les ressources, les protocoles multicast cherchent à construire un arbre reliant la source à chacune des destinations,et utilisent ensuite cet arbre pour écouler le trafic. Par rapport à une utilisation de plusieurs connections unicasts en parallèle, le multicast permet donc de réduire considérablement la redondance dans la transmission des données. Ainsi, le multicast semble être la solution la plus adaptée pour le transport simultanné de données entre plusieurs noeuds du réseau. Plusieurs mises en uvres pratiques du multicast ont été proposées pour les réseaux IP [1, 2]. Pourtant, les protocoles multicast sont encore assez peu déployés dans les réseaux actuels, où les transmissions sur l'Internet courant sont encore dominés par de l'unicast. La principale raison de cette faible pénétration des protocoles multicast réside dans la grande complexité de gestion et de mise en oeuvre des plans de routage. Mais, en raison de l'énorme croissance des demandes pour certaines applications massivement multi-utilisateur, telles que la visioconférence ou les jeux en ligne, de nombreux fournisseurs de services Internet (FSI) recommencent à envisager des solutions basées sur le déploiement de protocoles multicasts.

Contrairement aux applications évoquées ci-dessus et qui se caractérisent par des contraintes proches du temps réel, des applications, tel que la Vidéo à la Demande (VoD), qui sont également très populaires, ne nécéssitent généralement pas une diffusion simultannée des contenus à plusieurs clients. Un protocole de type multicast n'est donc pas adapté à la diffusion de contenus pour de tels services qui utilisent donc essentiellement de l'unicast. Rappelons que les services de type VoD permettent aux abonnés de regarder/écouter des vidéos ou des contenus audio à tout moment et avec en choisissant un niveau de qualité de service (QoS) en fonction du mode d'accès et du terminal. Le trafic générés par les plus populaires de ces contenus, représente aussi une part importante du trafic Internet mondial [3]. Le concept de CDN (*Content Delivery Network*) a émergé depuis peu de la communauté des principaux acteurs de l'Internet, afin d'améliorer la mise en œuvre des systèmes de diffusion de contenus, tels que la VoD. Dans un CDN, les contenus les plus populaires, parmis les objets Web (réseaux sociaux, graphiques), les objets de téléchargement (mises à jour logicielles), les contenus audios ou vidéos, sont stockés sur des serveurs qui sont installés à la périphérie des

réseaux, plus proches des clients. En plus de réduire les temps de transmission, ces systèmes contribuent également beaucoup à réduire le trafic dans le réseau coeur.

Les protocoles multicast et les systèmes de stockage distribué ont été largement étudiées. L'apparition du codage réseau en 2000, offre de nouvelles possibilités pour améliorer les performances réseaux, en le combinant avec du multicast et en l'utilisant dans un système de stockage distribuè. A l'origine, le codage réseau est une technique issue du domaine de la théorie de l'information, pour atteindre un débit théorique maximal dans un réseau multicast [5]. Analysé au moyen d'outils algébriques, le codage réseau apparait comme un schéma de codage générique qui permet de combiner des informations au niveau des nœuds intermédiaires d'un réseau. L'opération qui permet de combiner du trafic dans un seul flux est désigné sous le terme *codage* ou *encodage*, alors que le mécanisme qui permet de retrouver les informations originales au sein d'un flux codé s'appelle *décodage*. Ces opérations changent fondamentalement le schéma de routage traditionnel et elles offrent une autre façon de traiter les problèmes de congestion. Le premier bénéfice mis en évidence est l'amélioration des performances dans les réseaux multicast. Dans [5], les auteurs montrent que l'utilisation du codage réseau pour les communications multicast permet d'atteindre la capacité maximale en terme de débit dans le réseau. Dans cette thèse, nous utilisons le terme *La Capacité Réseau* pour désigner le débit maximal qui peut être atteint simultanément dans un réseau, entre une source et plusieurs terminaux. En outre, nous désignons par le terme *Le Seuil*, le débit maximal théorique obtenu lorsqu'on utilise du codage réseau. Il est montré dans [65] que le codage linéaire est déjà suffisant pour atteindre le seuil. Cependant, à ce stade, le codage réseau est encore irréaliste, car il nécessite de déterminer à l'avance les coefficients de codage sur tous les lien d'un réseau. Les auteurs dans [43] présentent une approche distribuée pour le codage réseau et montrent qu'il suffit de choisir les coefficients de manière aléatoire dans un corps fini judicieusement choisi pour obtenir un schéma de codage pour lequel la probabilité de parvenir à décoder toutes les informations est très élevée. Ces travaux montrent, pour la première fois, que le codage réseau peut être utilisé en pratique.

La recherche sur le codage réseau a reçu beaucoup d'attention durant ces dernières années. Des travaux ont notament porté sur les avantages du codage (en terme de sécurité, de robustesse, de fiabilité, de débit, etc.) dans les réseaux maillés et les réseaux sans fils. Pour une première introduction au codage réseau et une initiations à ces principales applications, nous renvoyons le lecteur au Chapitre

Dans cette thèse, nous définissons le bénéfice du codage par le gain de débit obtenu en utilisant le codage réseau (par rapport au débit atteint dans un réseau multicast traditionel). S'il est désormais bien connu que le codage réseau permet d'atteindre le débit maximal dans un réseau, la question d'évaluer le débit atteignable dans un réseau multicast, et donc de comparer les deux, est plus complexe. Depuis 2005, plusieurs travaux ont porté sur cette question particulière d'évaluer, en théorie et en pratique, les débits que l'on peut obtenir dans un réseau multicast avec ou sans codage réseau. Il est montré dans [46] que le bénéfice du codage réseau en terme de débit est théoriquement non borné dans des réseaux orientés. Ce résultat est fortement contre-balancé par les travaux de Li *et al* dans [67] qui montre que ce bénéfice est borné par 2 dans des graphes non-orientés. Plus tard, en 2012, Yin *et al* dans [87] ont montré que le bénéfice disparait tout à fait dans le cas des réseaux bi-orientés. De premières expériences numériques sont abordées dans [86] où des résultats obtenus sur six réseaux de FAI sont comparés. Aucun résultat ne montre un gain de débit lié à l'utilisation de codage réseau. Des travaux similaires dans [66] sur des instances de réseaux aléatoires et non-orientés donne le même résultat. On peut cependant relever certaines limitations quand à ces résultats, lié à la méthodologie d'évaluation utilisée. En effet, comme le calcul du débit optimal dans un réseau multicast se ramène à plusieurs évaluations du problème de l'arbre de Steiner et que ce problème est connu pour être NP-complet, les résultats présentés s'appuie, pour résoudre ce problème, soit sur l'utilisation d'algorithmes d'approximation, voire d'heuristiques, soit sur des approches énumératives exhaustives, donc forcément très limitées quant à la taille des instances traitables. En dépit de l'existence de ces limites, les travaux cités ci-dessus avaient le mérite de mettre en lumière certaines particularités du problème, à savoir, la sensibilité à la structure topologique de réseau qui pourrait impliquer que les instances de "réseau papillon" qui illustrent classiquement l'écart de débit entre multicast et network coding sont particulièrement atypiques et peu présentes dans les réseaux réels.

Bien que le gain en débit généré par l'utilisation du codage réseau dans les réseau multicast reste difficile à évaluer, le codage réseau procure d'autres avantages: un plan de routage optimisé dans un réseau utilisant du codage réseau est très simple à obtenir, là où le problème équivalent dans le cas d'un réseau multicast classique reste très complexe. De plus, pour atteindre des valeurs de débit proche du débit optimal, il faut souvent utiliser un grand nombre d'arbres multicast [13]. A partir

de maintenant et dans le reste de la thèse, le terme *codage réseau* sera toujours utilisé en référence à son utilisation dans un réseau multicast. Le terme *multicast* fera référence, quant à lui, aux mécanismes de routages traditionnels utilisant des arbres multicast. En l'absence d'indications contraires, nous nous intéresserons toujours au cas de sessions multicast issues d'une source unique. Comme déjà évoqué, le calcul du débit optimal pour un problème de routage particulier peut se modéliser comme un problème d'optimisation. Ainsi, le calcul du débit maximal (et le schéma optimal de routage associé) dans le cas d'un réseau multicast sans codage, se ramnène au problème de *Fractional Steiner Tree Packing*, un problème d'optimisation *NP-difficile* bien connu [47]. Cependant, le problème d'optimisation à considérer dans le cas d'utilisation du codage réseau peut être résolu en temps polynomial; en effet, il suffit de résoudre une série problèmes de flot maximum, entre l'unique serveur et chacun des terminaux. Le problème de flot maximum est un problème classique en théorie graphe et il existe de nombreux algorithmes efficaces pour le résoudre [28]. Certains travaux s'intéresse également à des problèmes dans lesquelles il y a un coût associé à l'utilisation des liens du réseau. Ces coûts peuvent, par exemple, modéliser l'utilisation de la bande passante ou les consommations d'énergie. Lun *et al* dans [70] proposent une approche décentralisée pour trouver les sous-graphes de coût total minimaux dans le cas du routage avec codage réseau. Notons que, pour atteindre le même objectif dans le cas du multicast, il faut connaitre la topologie complète du réseau. Il est généralement difficile de maintenir une telle connaissance de manière centralisé.

Pour les services multicast, le codage réseau comme une technique avancée permet aux nœuds intermédiaires pour faire des calculs. Les fonctionnalités supplémentaires nécessitent le soutien de mise à jour logicielle ou matérielle sur la source et les terminaux ainsi que les nœuds de transition. Ces changements peuvent perturber l'autre trafics de données qui passent sur le même réseau, mais ils ne demandent pas le service de multicast. De plus, les opérations algébriques, comme l'encodage et le décodage, introduirent des charges sur les travails supplémentaires dans les nœuds d'un réseau. En conséquence, ils vont ralentir l'efficacité du traitement des données. Ces effets négatifs apportés par le codage réseau sont les préoccupations majeures pour les FAI d'appliquer cette nouvelle technique en pratique. Quelques études donc ont porté de chercher des façons pour réduire les interférences du codage réseau. Lucani *et al* dans [68] visaient à limiter le volume de flux du codage réseau afin de réduire les charge de calcul sur les nœuds qui faisaient l'encodage et le décodage. Ils proposaient un protocole de routage hybride

ainsi que son cadre de l'optimisation, où le codage réseau n'est considéré comme l'auxiliaire de multicast dans les transmissions des données. Dans ce problème, le coût minimum problème *d'arbre Steiner* pour les flux non-codés est toujours résolu par des algorithmes sous-optimal. Dans [53] et [55], un *génétique Algorithme* basé sur le cadre algébrique était créé pour trouver le nombre minimum des nœuds atteignant un débit donné et visant à minimiser la charge de mise à jour sur un réseau. Le problème est NP-difficile. Dans cet article, les auteurs constatent que, en général, un très petit ensemble des nœuds codages est déjà suffisante pour fournir le débit maximum.

Le codage réseau apporte des bénéfices non seulement dans les réseaux multicast, mais également dans les réseaux sans fils, les réseaux optiques, et les sysèmes de stockage distribués. La seconde partie de cette thèse se concentre sur l'application du codage réseau dans les systèmes de stockage distribués. En effet, dans cette application, le codage réseau montre une possibilité d'améliorer les performances réseaux au-delà de la considération de routage. Un document récent [4] a étudié un système de stockage modifié qui stocke les informations codés d'un contenu original. Un élèment ou un bloc d'information codé est constitué par des combinaisons linéaires aléatoires de tous les blocs d'un contenu original. Lorsque un utilisateur d'Internet visitent un tel système pour y chercher certains contenus, ils recevront des blocs d'informations codés, la taille totale de ce qu'ils recoivent étant égale ou légèrment supérieure à la taille du contenu d'origine. L'utilisateur peut ensuite récupérer le contenu original en décodant l'information reçue. Dans [4], il est démontré que ce nouveau système est très efficace pour la transmission des données: d'une part, les transmissions ont une très faible latence, et, d'autre part, la probabilité de succès pour le décodage de l'informations est grande. L'enquête menée dans [20] montre que l'utilisation du codage dans les systèmes de stockage distribués améliore la fiabilité du système. Les auteurs dans [27] soulignent que l'utilisation du codage réseau dans les systèmes de CDNs permet de réduire l'utilisation des serveurs et donc également, la consommation d'énergie. Ceci s'explique par le fait que l'application du codage linéaire aléatoire réduit la probabilité de blocage lorsque la même information est accédée par des terminaux différent simultanément. Une étude dans [38] se concentre sur les applications du codage réseau dans l'allocation de stockage pour transmettre des contenus vidéos dans des réseaux sans fils. Cette étude montre que le système de stockage utilisant le codage réseau facilite la modélisation mathématique. De plus, le système de stockage correspondant fournit de meilleures performances pour le téléchargement de fichiers. Les auteurs dans [62, 63, 64] posent les bases

d'une analyse de la probabilité de trouver l'allocation optimale offrant une grande fiabilité pour les utilisateurs cherchant à décoder les informations codées. Il serait intéressant d'étudier également le problème mixte de stockage et de routage dans un cadre d'optimisation, car il peut être considéré comme une extension de problème classique de transport.

Motivations et contributions

Confortés par l'ensemble de ces observations, nous croyons qu'il est encore trop tôt pour affirmer que le multicast peut être systématiquement renforcés par du codage réseau. Le postulat théorique initial de gain en débit reste encore difficile à appréhender, surtout parce que le problème de maximisation de débit dans un réseau multicast est difficile à résoudre. De plus, nous ne pouvons pas ignorer les considérations pratiques liées au déploiement du codage réseau, tels que les mises à jour nécessaires de certains équipements et les activités d'encodage et de décodage. En particulier, si la quantité d'informations codées est importante, la complexité du décodage peut devenir problématique. La plupart des terminaux utilisés par les clients, comme, par exemple, la Livebox, les tablettes ou les téléphones portables, ne sont généralement pas capables de réaliser des calculs trop complexes. Il est, par conséquent, essentiel d'évaluer soigneusement les avantages et les inconvénients apportés par le codage réseau, y compris la charge de calcul supplémentaire. De plus, différentes stratégies de stockage peuvent avoir un impact significatif sur le comportement de routage. Il est donc utile d'étudier le problème de routage dans les systèmes de stockage distribués utilisant le codage réseau.

Dans cette thèse, nous proposons d'abord, dans le Chapitre 3, une manière efficace pour calculer le débit multicast maximal ainsi que différentes variantes du problème. Nous résumons les problèmes de flot réseau, puis nous étudions les relations entre les problèmes de goulot d'étranglement (bottleneck) dans les arbres Steiner et les problèmes de débit maximal utilisant des arbres multicast. Certains résultats préliminaires concernant les problèmes de goulot d'étranglement sont également rappelés. La contribution principale de ce chapitre est que nous fournissons deux algorithmes en temps polynomial sur le problème de calcul du goulot d'étranglement lorsque l'on cherche relier des terminaux au moyen d'un arbre (arbre de Steiner) avec la contrainte additionnelle que chaque terminal est

une feuille de l'arbre (full bottleneck Steiner tree). Ces algorithmes peuvent être facilement mis en œuvre, car ils font appel à des concepts très simples de la théorie des graphes.

Dans le Chapitre 4, nous nous intéressons au problème de l'évaluation du bénéfice apporté par le codage en terme de débit. Nous proposons des modèles mathématiques et des algorithmes afin de maximiser le débit multicast et observons que nos approches sont suffisamment efficaces pour résoudre à l'optimalité des instances de problème de tailles moyennes, voire grande. Nous traitons également le problème de maximisation de débit multicast avec la contrainte additionnelle de n'utiliser qu'un nombre limité d'arbres, et pour ce problème, nous sommes obligé de nous limiter à de plus petites instances. Nous utilisons un outil commercial (XPress Optimizer Version 21.1.00) pour résoudre les problème linéaire (LP) ainsi que les problèmes en variables mixtes (MIP). Nous avons mené plusieurs séries de tests numériques sur des réseaux orientés et bi-orientés. Ces instances sont générées aléatoirement en utilisant notre propre générateur de graphes. Le premier résultat surprenant est que, sur l'ensemble des instances générées, orientés et bi-orientés, nous ne trouvons pas un seul réseau où le codage réseau (NC) ait un débit maximal plus grand que multicast (MC). Cependant, lorsque nous considérons le multicast avec un nombre limité d'arbres (MC-ℓ), dans lequel ℓ indique le nombre d'arbres utilisés, le résultat est très différent. Figure 1 montre les valeurs de débit relatif (100% signifie que le rapport de débit entre NC et MC est égal à 1) quand nous restreignons le nombre d'arbres à 1, 2 ou 3 pour multicast. Nous constatons que le rapport diminue lorsqu'on diminue le nombre d'arbres, ce qui signifie que la valeur du débit multicast diminue lorsqu'on diminue le nombre d'arbres. Les tendances générales sont très similaires pour les instances orientés et bi-orientés. La réduction de débit est beaucoup plus grande lorsque les instances sont plus denses (environ $6n$ liens pour l'ensemble des premières instances et $3n$ liens pour les autres, où n représente le nombre des nœuds dans le réseau). Cela est dû au fait qu'un nombre limité d'arbres ne permet pas d'exploiter pleinement la capacité potentielle offerte par le réseau alors que le codage réseau y parvient beaucoup mieux. Dans les réseaux de télécommunication traditionnels, le degré moyen est généralement assez faible (par exemple entre 3 et 5). Les observations sur la série des quatre derniers exemples montrent qu'il y a encore une réduction significative de débit (de 13% à 25%) lorsqu'on utilise jusqu'à 3 arbres multicast, par rapport à une solution du codage réseau. Il ressort de ces expérimentations que le codage réseau peut être considéré par les administrateurs réseaux comme une alternative très intéressante aux solutions standards de routage.

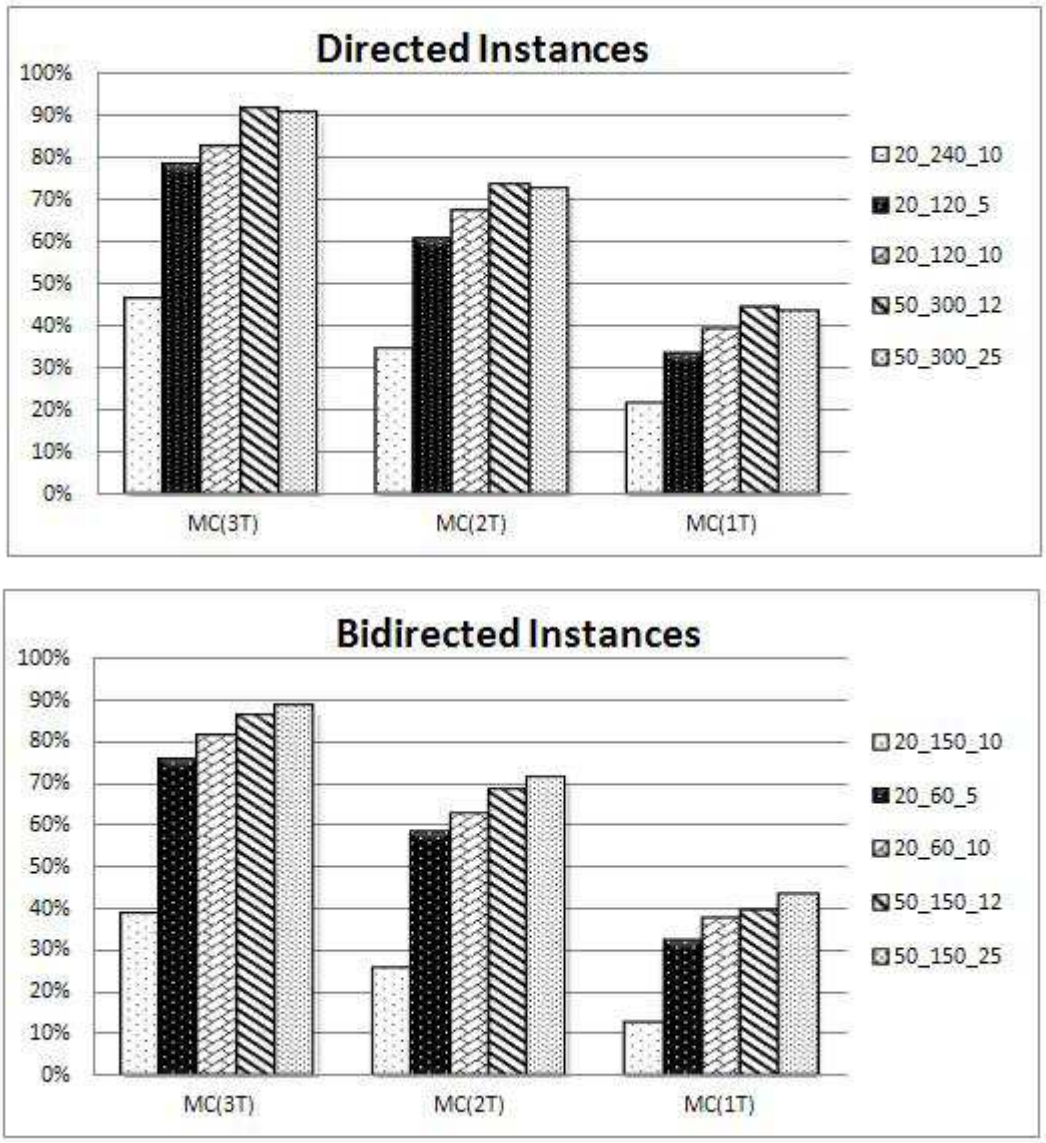


Figure 1: Comparaison des débits multicast sans (MC) et avec codage réseau (NC): la légende, par exemple 20_240_10, signifie un graphe généré aléatoirement, avec 20 nœuds, 240 arcs et 10 parmi les 20 nœuds sont des terminaux (Ce type légende sera utilisé dans le reste de ce chapitre). Les hauteurs des colonnes représentent les débits relatifs qui sont atteint par MC avec 1, 2 et 3 arbre(s) sur les différents ensembles des instances. De plus, les valeurs correspondantes sont les moyennes sur 100 instances générées aléatoirement pour chaque type. Le maximum (100%) correspond au débit NC.

Figure 2 donne un aperçu sur le nombre d'arbres multicast nécessaires pour atteindre le même débit que le codage de réseau fait. Les chiffres obtenus dans les boîtes compte 50% des instances générés. Nous observons que la variance

de chaque type est assez élevé, ce qui indique que certain cas requiert seulement un petit nombre d'arbres à atteindre le débit maximum mais certain cas nécessitant un grand nombre. Les valeurs diminuent lors de l'augmentation du nombre de terminaux. Cela peut être dû au fait que, grâce à la théorème de Edmonds d'emballage arborescence [25], l'avantage de codage s'annule lorsque tous les nœuds sont terminaux. Si cinq arbres serait considéré comme une limite supérieure raisonnable pour les opérateurs à manipuler, et puis, dans la plupart des cas, le débit NC n'aurait pas être réalisé par le multicast. D'autre part, dix arbres peuvent souvent suffire pour les petits réseaux.

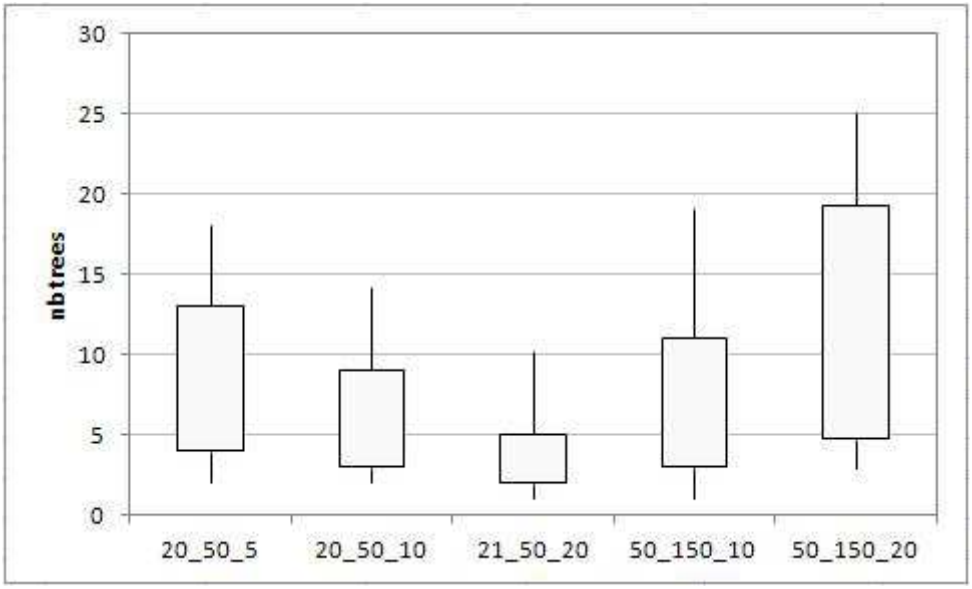


Figure 2: Nombre d'arbres nécessaire pour atteindre le débit optimal: pour chaque groupe (indiquée sur l'axe des abscisses), 100 instances aléatoires sont générées et le nombre minimal d'arbres multicast nécessaires pour obtenir le débit optimal est calculé: 50% des cas se situent dans les boîtes et 90% se situent dans les intervalles.

La Figure 3 montre la moyenne, sur les séries indiquées, des temps pour calculer les débits optimaux en utilisant nos modèles. Nous voyons clairement que les calculs des débits multicast prennent généralement quelques minutes tandis que les calculs des débits NC sont instantannés. Les calculs pour un seul arbre multicast sont aussi très rapide, mais pour les nombres supérieures à 2, les problèmes deviennent plus difficiles à résoudre en pratique.

En se concentrant de manière plus approfondie sur les instances de petites tailles, on s'aperçoit que toutes les instances uniformes (toutes les capacités dans le réseau sont égales à 1) générés de 7 à 10 nœuds ont le même débit multicast que NC (cf.

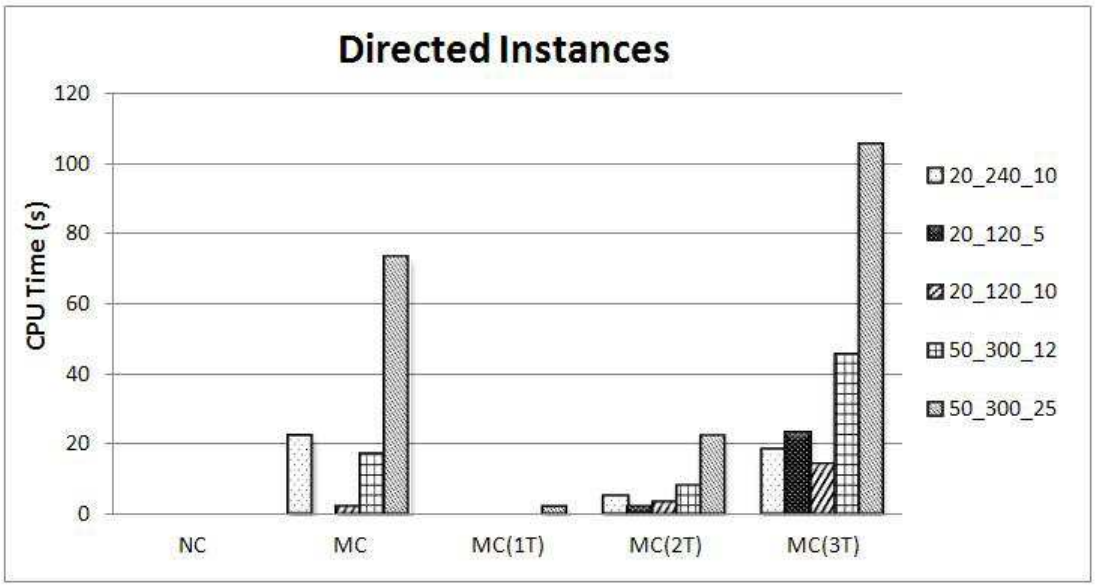


Figure 3: Temps de calcul moyens (en secondes) pour résoudre les problèmes de maximisation de débit.

Tableau 1). En particulier, cela signifie que notre générateur ne parvient pas à reproduire le réseau 'classique' en forme de papillon.

Pour essayer de mieux cerner ce phénomène surprenant, nous avons réalisé une recherche exhaustive dans tous les graphes avec 7 nœuds, 1 source et 2 terminaux, et toutes les capacités égales à 1. Pour limiter la recherche, nous considérons seulement les cas où au moins deux arcs sont issus de la source et au moins deux arcs entrent dans chaque terminal. Parmi tous les 950 951 instances possibles (en supprimant 18 016 instances non-connexes), seulement 96 instances montrent des écarts non nuls entre NC et MC. En fait, ces 96 cas peuvent tous se ramener seulement aux 3 instances decrites dans Figure 4, tous les autres cas étant symétriquement équivalents à ces 3 cas. Le premier est le réseau papillon, tandis que les deux autres sont juste des petites variantes autour de ce réseau. Si nous considèrons une distribution uniforme pour générer toutes les instances, alors la probabilité d'avoir un graphe avec un écart non nul est d'environ 0,01%.

Une étape ultime dans cette voie de recherche consiste, au lieu d'énumérer toutes les instances possibles d'un certain type, à considér le problème qui consiste à calculer (au moyen d'un modèle d'optimisation) une instance où l'écart est maximum. Comme ce problème s'est avéré très difficile à résoudre, nous nous sommes

Set	n	m	ks	kt	λ_{NC}	λ_{MC}	# abres
Instances n_rand_rand							
AVG	7	14,35	1	4,03	2,95	2,95	4,83
MIN	7	8	1	2	1	1	1
MAX	7	21	1	6	6	6	30
AVG	8	18,24	1	4,52	3,25	3,25	6,15
MIN	8	9	1	2	1	1	1
MAX	8	28	1	7	7	7	42
AVG	9	22,71	1	5	3,54	3,54	7,3
MIN	9	10	1	2	1	1	1
MAX	9	36	1	8	8	8	56
AVG	10	27,81	1	5,43	3,9	3,9	9,08
MIN	10	11	1	2	1	1	1
MAX	10	45	1	9	9	9	72

Table 1: Résultats sur des graphes aléatoires avec des capacités uniformes: les résultats sont les moyennes sur 1000 instances générées aléatoirement. Chaque instance contient une source et des liens avec des capacités uniformes $C_a = 1, \forall a$.

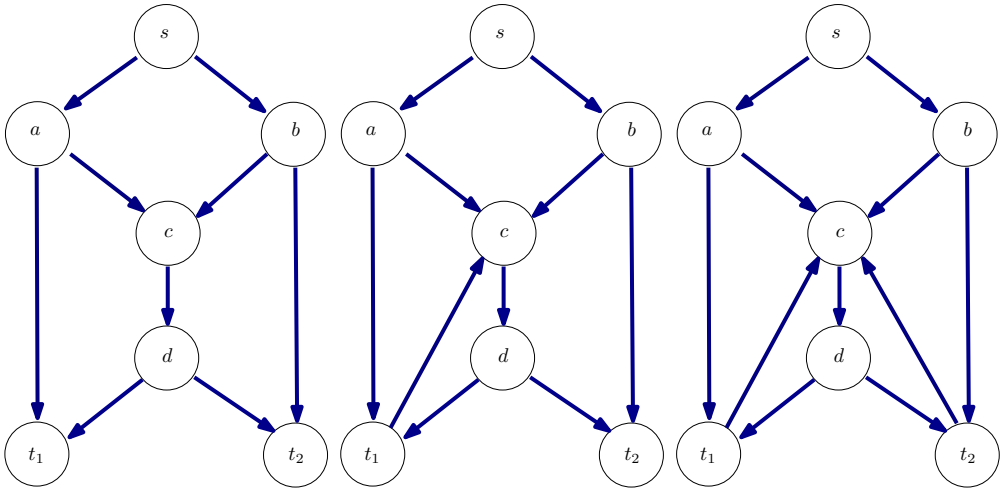


Figure 4: Les seules 3 instances (uniformes) avec un écart non nul (de 0,5) entre le débit NC et MC

limité à des cas de très petites tailles (voir le Tableau 2). Il est intéressant de noter que, pour les instances uniformes avec 7 et 8 nœuds, les seuls cas avec des écarts de débits non nuls entre NC et MC sont ceux ayant 7 nœuds et 2 terminaux, ou bien 8 nœuds et 2 ou 3 terminaux. Le premier cas correspond à nouveau au réseau papillon. Deux graphes pour les instances de 8 nœuds et de 2 ou 3 terminaux sont représentées sur la Figure 5. Il est facile de vérifier, sur ces deux graphes, que les écarts sont bien de 0,5.

Nous venons donc de montrer de façon expérimentale le résultat suivant:

n	k	c_a	$\Delta^*(NC, MC)$	λ^*_{NC}	λ^*_{MC}	m	cpu (sec)
7	2	1	0,5	2	1,5	9	0,8
7	3	1	0	-	-	-	0,2
7	4	1	0	-	-	-	0,1
7	5	1	0	-	-	-	0,1
7	6	1	0	-	-	-	0,1
8	2	1	0,5	3	2,5	13	380
8	3	1	0,5	2	1,5	11	12
8	4	1	0	-	-	-	0,5
8	5	1	0	-	-	-	0,1
8	6	1	0	-	-	-	0,1
8	7	1	0	-	-	-	0,1

Table 2: Maximisation de lécarts en débit entre MC et NC sur des instances uniformes.

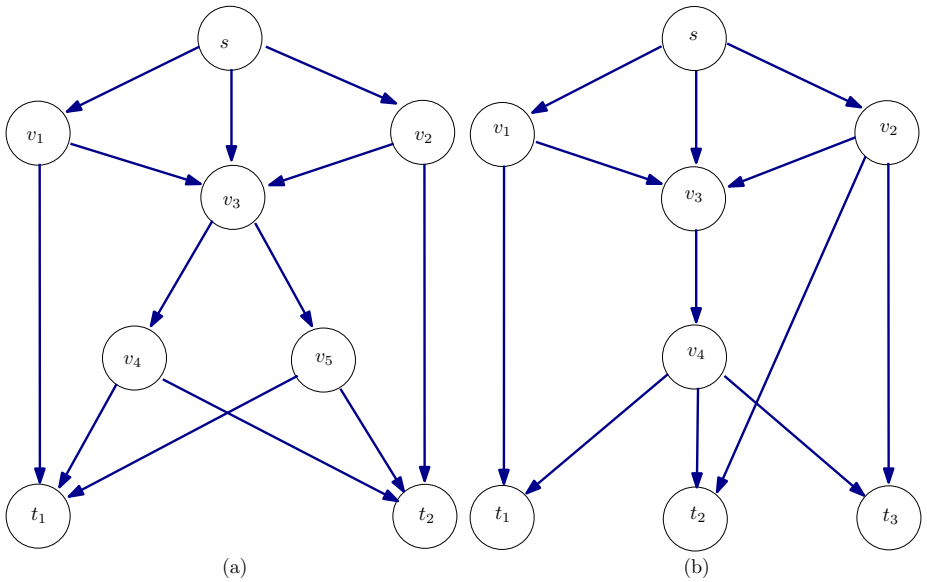


Figure 5: les deux instances uniformes à 8 nœuds avec un écarts de 0,5 entre NC et MC.

Lemma. *On considère un réseau orienté à n nœuds ($n = 7$ or 8), une source unique et k terminaux (différent de la source). On suppose que les capacités sont toutes égales à un: $C_a = 1, \forall a \in A$. Si $n = 7$ et $k \geq 3$ ou $n = 8$ et $k \geq 4$, le codage réseau n'améliore pas le débit par rapport au multicast standard.*

En conséquence, nous confirmons que, sauf pour certains graphes très particuliers, l'avantage apporté par le codage réseau dans la transmission multicast est relativement faible, et ce, même dans le cas des graphes orientés, pour lesquels le gain en débit est prétendument illimité. En effet, les topologies particulières ex-

hibants un écart de débits entre le codage réseau et le multicast (réseau papillon) sont tellement spécifiques qu'elles ne se rencontrent presque jamais en pratique. Pourtant, comme le multicast requiert presque toujours un nombre élevé d'arbres pour obtenir un débit élevé, le codage réseau reste une alternative tout très attractive pour la gestion des futurs réseaux.

Les opérations algébriques nécessaires à l'encodage et au décodage des trames d'informations, génèrent une charge de calcul supplémentaire au niveau des équipements de routage, ce qui peut avoir pour effet d'induire des délais supplémentaires dans la diffusion des information. Les résultats exposés dans le Chapitre 4 ont montré que l'utilisation du multicast induit aussi certaines limitations. C'est pourquoi nous avons décidé d'étudier, et si possible d'optimiser, la façon de mettre en oeuvre les protocoles de routage, et ce, afin de réduire les effets négatifs soit du codage réseau, soit du multicast pris isolément l'un de l'autre. Dans le Chapitre 5, nous considérons d'abord le problème de minimiser le nombre des nœuds faisant l'encodage et d'évaluer le compromis entre la duplication multicats et le codage aux niveaux des nœuds intermédiaires. Nous avons utilisé deux approches différentes. Dans la première, chaque demande de trafic pour un service multicast est réparti, selon un rapport fixe $\alpha \in [0, 1]$, entre un arbre de diffusion multicast et plan de routage basé sur du codage réseau. Dans la seconde approche, le nombre de nœuds où sont réalisées les fonctions de réplication multicast ou d'encodage est limitée par un nombre fixé à l'avance (que l'on peut interpréter comme issue d'une contrainte budgétaire). Dans chaque cas, nous construisons un modèle d'optimisation permettant de calculer le débit maximum atteignable.

Les instances sont générées aléatoirement (de la même façon que dans Chapitre 4) par construire un graphe fortement connexe (avec un chemin entre la source et chaque terminal), puis augmenter progressivement la densité du graphe jusqu'à un niveau requis. Les capacités d'arcs sont générées aléatoirement dans l'intervalle $[0, 10]$. Dans les groupes de données (n, m, n_T), n, m, n_T représentent le nombre de nœuds, de l'arcs, des terminaux, respectivement. Des séries de 100 cas sont générés aléatoirement et les résultats moyens sont rapportés.

Plusieurs expérimentations de la littérature ont montré qu'une stratégie de routage utilisant plusieurs arbres multicast permet d'atteindre, ou d'approcher, le débit du codage réseau, mais le nombre d'arbres multicast nécessaires peut être très grand. Comme une telle approche n'est pas envisageable en pratique, nous con-

sidérons ici le cas où un seul arbre multicast est utilisé. Dans ce cas, comme notre première série d'expériences l'a confirmé, il y a, la plupart du temps, un écart énorme entre les débits multicast et codage réseau. La Figure 6 montre les débits relatifs obtenu en utilisant des stratégies intermédiaires utilisant un ou deux arbres multicast et du codage réseau:

- NC+BC: mélange de codage réseau et de broadcast (sur certains nœuds);
- NC(p=1)+BC: idem que NC+BC mais avec un seul nœud réalisant de l'encodage;
- NC(p=2)+BC: idem que NC+BC, mais avec deux nœuds réalisant de l'encodage;
- MC(1t): multicast utilisant un seul arbre.

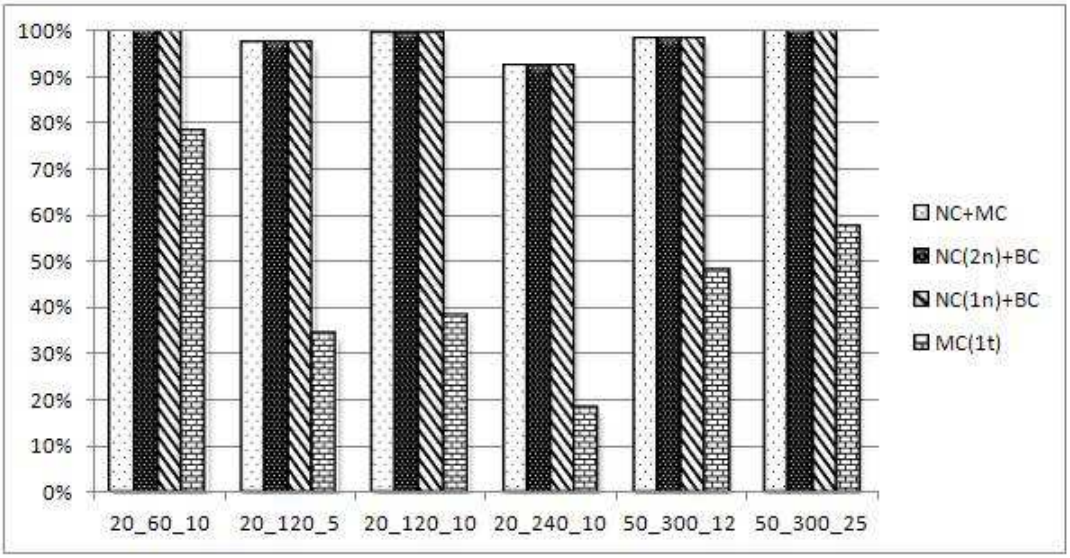


Figure 6: Débits relatifs obtenus avec les différentes stratégies de codage/routage

On peut faire les observations suivantes: le débit obtenus avec un seul arbre multicast MC(1t) est toujours bien en dessous du débit maximum atteint par le codage réseau. En mélangerant le broadcast (BC) (au lieu de la réplication partielle, c'est-à-dire MC) avec du codage réseau (NC), le débit est très proche du débit optimal. Dans le cas des instances les plus denses (par example 20_240_10), la réduction du débit peut atteindre près de 10%, mais ces cas sont très éloignés des topologies typiques des réseaux de télécommunication. Enfin, et c'est peut-être l'observations la plus surprenante, dans presque tous les cas, un seul nœud faisant l'encodage est suffisant pour atteindre le débit optimal. Cela milite beaucoup en

faveur du codage réseau parce que le coût de déploiement de la fonctionnalité d'encodage sur un seul nœud est très faible.

Afin de réaliser une analyse plus fine, nous comparons l'impact du nombre de nœuds faisant l'encodage avec un autre paramètre important, à savoir la capacité totale sur tous les liens entrants dans chaque terminal, qui constituent souvent les goulets d'étranglement dans les réseaux télécommunications. Dans notre modèle, nous considérons plutôt le degré entrant sur chacun terminal, que nous limitons à certains valeurs (1, 2 ou 3 dans nos expériences).

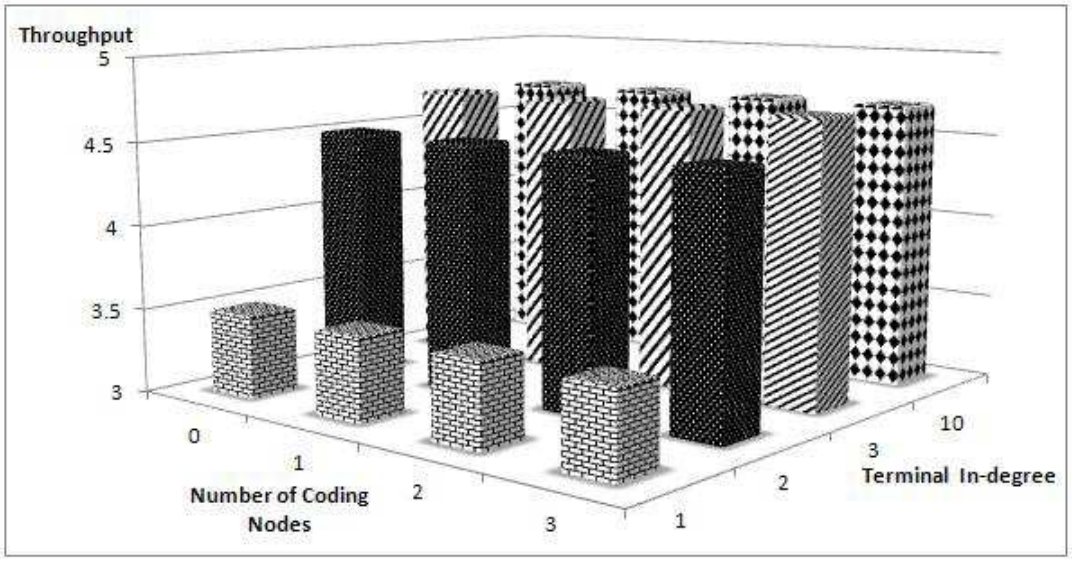


Figure 7: Moyenne des débits maximaux obtenus sur des instances 20_60_10 lorsque les nœuds faisant l'encodage et les degrés entrants aux terminaux sont limitées.

On voit de manière évidente sur la Figure 7 que le nombre de nœuds faisant l'encodage a (encore) un impact très limité, tandis que la limitation du flux entrant dans les terminaux a un impact plus important.

Les courbes de la Figure 8 donnent une indication sur les débit que l'on peut obtenir en combinant du routage multicast pur avec un plan de routage utilisant du codage réseau. Nous voyons que, hormis pour les plus denses (20_240_10), il est déjà avantageux d'utiliser le codage réseau pour un petit pourcentage du trafic. On observe, par exemple, une amélioration du débit relatif de 4% lorsque 10% du trafic utilise du codage réseau et cette augmentation est plus ou moins

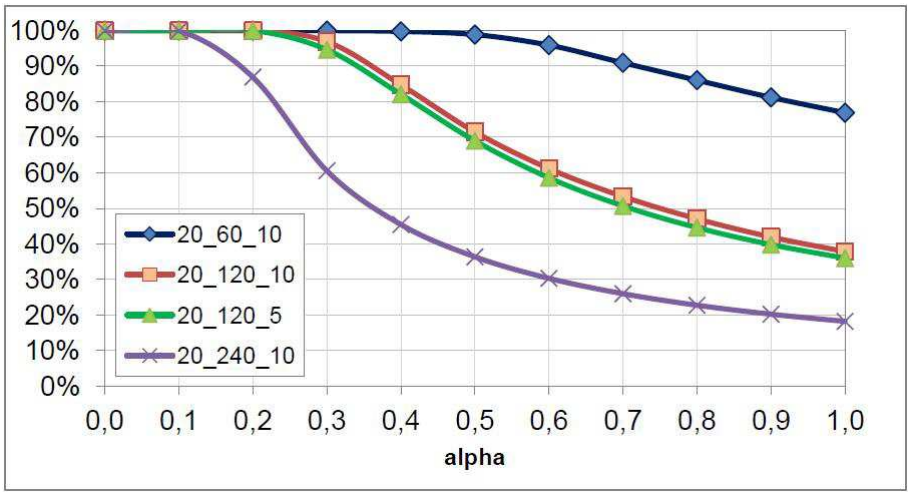


Figure 8: Pourcentage du débit optimal obtenu en routant α% du trafic sur un arbre multicast (et les $(1-\alpha)$% restant selon un plan de routage utilisant du codage réseau).

linéaire jusqu'à atteindre le débit maximal (obtenu lorque ce trafic atteint une fourchette entre 50% à 70% du trafic total). Par conséquent, si le rapport entre le coût et le bénéfice du codage réseau est de moins de 1 sur 2, il semble être intéressant de déployer du codage réseau.

`(Eric): I am not sure I am following you here ???`

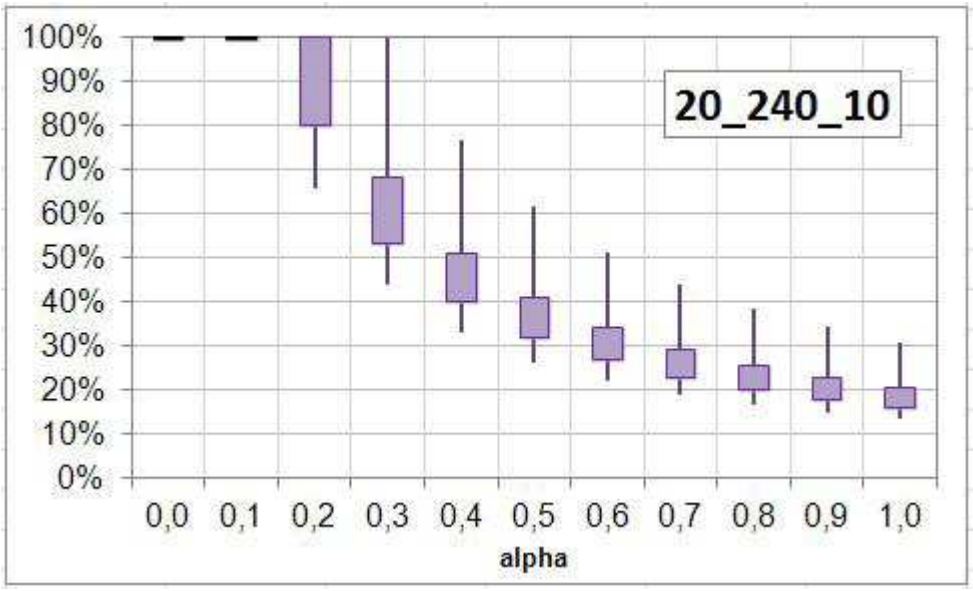


Figure 9: Répartition des valeurs autour de la moyenne pour le pourcentage de débit obtenu en routant α% du trafic sur un arbre multicast: les boîtes représente 50% des cas, les valeurs extrêmes $[\min, \max]$ sont représentées par les segments.

La Figure 9 permet d'analyser plus en détails le cas le plus dense (20_240_10), en indiquant l'étalement des valeurs observées autour de la moyenne. Si la gamme des valeurs extrêmes est assez grand, nous pouvons observer que 50% des instances suivent la tendance annoncée par la valeur moyenne, à savoir que l'impact de l'introduction du codage réseau augmente de manière considérable lorsque plus d'un tiers du trafic est transporté avec du codage réseau.

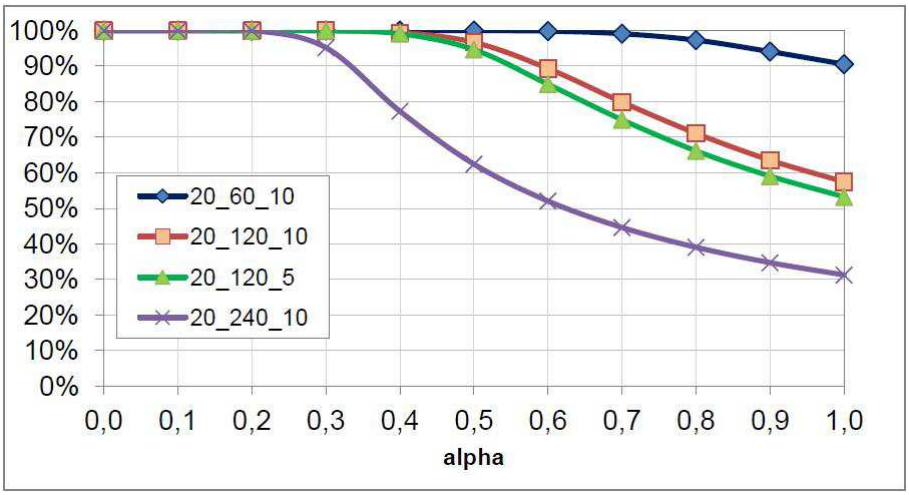


Figure 10: Pourcentage du débit optimal obtenu en routant α% du trafic sur deux arbres multicast (et les $(1-\alpha)$% restant selon un plan de routage utilisant du codage réseau).

La Figure 10 donne les résultats obtenus lorsque l'on s'intéresse au cas où deux arbres multicast sont utilisés en parallèle avec du codage réseau. On suppose ici que la proportion α% du trafic géré en multicast est également répartie sur les deux arbres ($\alpha/2$% sur chaque arbre). Nous voyons que la différence entre le débit du codage réseau ($\alpha = 0$) et le débit multicast ($\alpha = 1$) est réduite par rapport au cas dans Figure 8, mais les tendances des courbes restent les mêmes: par exemple, l'introduction d'une petite quantité du codage réseau est déjà avantageuse, mais l'avantage doit être comparé au coût engagé. Pour obtenir une performance presque optimale, il faut que plus de 50% du trafic soit géré par du codage réseau. On observe néanmoins que cette proportion tombe à 30% dans le cas des réseaux les moins denses (20_60_10).

Pour résumer, nos résultats (essentiellement obtenus sur des instances générées aléatoirement) montrent qu'un petit nombre de nœuds faisant de l'encodage est suffisant pour atteindre la capacité du réseau; les gains en débit obtenu par l'utilisation du codage réseau augmentent considérablement dans les réseaux

denses; l'information dupliquée sur les liens sortants joue un rôle important dans le gain en débit. Nous avons observé que l'introduction d'une petite proportion de codage réseau dans le volume des flux multicast produit déjà une augmentation significative du débit global. Cependant, pour obtenir un débit encore plus élevé lorsqu'on utilise de manière conjointe un plan de routage géré par du codage réseau et du routage multicast traditionnel, une partie relativement importante (environ 30% pour les graphes le plus clairsemés jusqu'à 50% pour les graphes denses) du trafic doit être transporté par du codage réseau pour atteindre un débit presque optimal.

Dans une deuxième partie de la thèse, nous avons étudié un problème de transport sur les systèmes de stockage distribué utilisant le codage réseau pour stocker l'information. En fait, le schéma de codage supprime l'importance de la pièce unique d'information dans un système, et par conséquent, il permet une plus grande flexibilité pour le stockage et améliore les temps d'accès aux données par les clients. En effet, il suffit de garantir que les clients recoivent la même quantité d'informations codées que les informations d'origine pour que le décodage puisse se faire. En conséquence, les nouvelles stratégies de placement de l'information ont ausso un impact sur les schémas de routage nécessaires à la diffusion, en particulier lorsque l'accès aux serveurs et/ou aux terminaux des clients deviennent les goulots d'étranglement du système. Nous avons étendu notre modèle d'optimisation à un problème d'optimisation plus général, mais qui n'a pas été, à notre connaissance, étudié dans la littérature, à savoir, le problème de transport avec des contraintes degrés. Nous proposons une méthode de résolution pour ce probème basé sur la décomposition lagrangienne.

En résumé, les contribution principales de cette thèse sont de fournir des modèles mathématiques et des algorithmes efficaces pour calculer le débit optimal lorsqu'on utilise différents plan de routage combinant le multicast traditionnel avec du codage réseau. Cela nous a permis de clarifier, au travers de nombreux test numériques, l'avantage du codage réseau. En particulier, nous avons démontrer les avantage d'un routage hybride, qui apporte, non seulement un gain significatif en débit, mais s'avère également très simple et peu invasif en terme de nouvelle fonctionnalités à déployer. En outre, notre étude d'un système de stockage distribué utilisant le codage réseau nous a permis d'aborder un problème nouveau, à savoir un problème de transport avec contraintes degrés.

Organisation

La thèse est organisée comme suit. Dans le Chapitre 2, nous donnons un bref aperçu des principes du codage réseau, des réseaux multicast, et les systèmes de stockage distribué. Nous y introduisons également les outils méthodologiques utilisés durant la thèse, à savoir, des outils de la théories des graphes et des modèles classiques d'optimisation dans les réseaux de télécommunication.

Le corps principal de la thèse est séparé en deux parties. La première partie, qui est la plus importante, inclue les Chapitres 3, 4 et 5. Cette partie concerne l'étude des modèles mathématiques et des algorithmes efficaces permettant de résoudre différents problèmes de calcul de débit maximal dans les réseaux et à évaluer l'avantage du codage et de nouvelles techniques de routage hybride. La deuxième partie est developpée dans le Chapitre 6, et donne les premiers résultats d'une étude sur le problème de transport dans des systèmes de stockage distribué utilisant le codage réseau.

Dans le Chapitre 3, nous proposons des algorithmes efficaces pour résoudre les problèmes de calcul de débit maximal lors de l'utilisation d'un ou de plusieurs arbres multicast. Ces modèles servent de base aux études menées dans le Chapitre 4 et qui revisite les problématiques d'évaluation du gain de débit entre le codage réseau et le multicast. Nous proposons un algorithme heuristique simple pour évaluer le débit maximum obtenu par un routage qui utilise un nombre limité d'arbres multicast. Dans le Chapitre 5, nous fournissons les résultats d'une étude numérique intensive qui nous a permis d'évaluer le compromis entre l'utilisation du codage et la duplication aux nœuds intermédiaires, ainsi que des nouveaux schémas de routages hybrides. Dans le Chapitre 6, nous étudions un problème de transport avec des constraints de degrés, qui permet de modéliser des problèmes de routage statique dans les systèmes de stockage distribués qui utilisent le codage réseau.

Enfin, nous résumons nos études et proposons des extensions possibles des travaux dans le Chapitre 7.

Abstract

The popularity of the great variety of Internet usage brings about a significant growth of the data traffic in telecommunication network. Data transmission efficiency will be challenged under the premise of current network capacity and data flow control mechanisms. In addition to increasing financial investment to expand the network capacity, improving the existing techniques are more rational and economical. Various cutting-edge researches to cope with future network requirement have emerged, and one of them is called network coding. As a natural extension in coding theory, it allows mixing different network flows on the intermediate nodes, which changes the way of avoiding collisions of data flows. It has been applied to achieve better throughput and reliability, security, and robustness in various network environments and applications. This dissertation focuses on the use of network coding for multicast in fixed mesh networks and distributed storage systems. We first model various multicast routing strategies within an optimization framework, including tree-based multicast and network coding; we solve the models with efficient algorithms, and compare the coding advantage, in terms of throughput gain in medium size randomly generated graphs. Based on the numerical analysis obtained from previous experiments, we propose a revised multicast routing framework, called strategic network coding, which combines standard multicast forwarding and network coding features in order to obtain the most benefit from network coding at lowest cost where such costs depend both on the number of nodes performing coding and the volume of traffic that is coded. Finally, we investigate a revised transportation problem which is capable of calculating a static routing scheme between servers and clients in distributed storage systems where we apply coding to support the storage of contents. We extend the application to a general optimization problem, named transportation problem with degree constraints, which can be widely used in different industrial fields, including telecommunication, but has not been studied very often. For this problem, we derive some preliminary theoretical results and propose a reasonable Lagrangian decomposition approach.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The most beautiful thing we can experience is the mysterious. It is the source of all true art and all science. He to whom this emotion is a stranger, who can no longer pause to wonder and stand rapt in awe, is as good as dead: his eyes are closed.

-Albert Einstein (1879-1955)

1.1 Background

According to a white paper from Cisco [3], the global IP traffic will increase threefold by 2016; especially, the growth of some particular traffic such as Video-on-Demand (VoD), live Internet TV or on-line gaming, is expected to be spectacular over the next five years. As a result, network operators will have to make significant financial investments in order to increase network capacities to satisfy the future demands with existing technologies. However, improving data transmission techniques, such as finding better solutions to reduce the volume of concurrent network traffic, or changing the data collision avoidance mechanisms, appears to be a more rational and economical solution.

The most important characteristic of services like live Internet TV or on-line gaming is that their data are always accessed simultaneously by a large amount of Internet users. When these applications employ unicast or broadcast, which are widely used in today's telecommunication networks, a considerable amount of duplicate network traffic has to be sent over networks, wasting a lot of network

resources. Unicast is the simplest and the most frequently used routing paradigm. The transmission is set up between single source and terminal to exchange information. If multiple terminals require the same service from a single source, multiple unicast sessions will be established at the same time, therefore the data will be duplicated and transferred through the network. Broadcast is a routing mode in which a network node copies the incoming information and forwards it to all his neighbor nodes connected by a link or in an effective communication range in a wireless network. It is usually called flooding technique. Although it guarantees the data will finally be delivered to every terminal, there will be a great deal of information sent on the relevant nodes and the network resource might exhaust. In order to reduce the data flow when running the network applications that involve many users at the same time, a transmission mode called multicast was initially introduced in late 1980s for the case when a single server expects to communicate simultaneously with a set of clients. This routing scheme uses a tree-based forwarding strategy. Compared to multiple independent paths tactic for unicast, it can considerably reduce the redundancy during data transmission. As such, multicast seems to be a viable alternative to unicast and broadcast for the network applications with massive participants. Several practical implementations of the multicast paradigm have been proposed for the IP networks [1, 2]. However, very few multicast services have been deployed. At present, the transmissions for Internet services are still dominated by unicast. The complexity for implementing multicast is high, and it is one of the main reasons that slows down the application of multicast services. Owing to the rapid growth of the demands on some applications, such as videoconferencing and massive on-line multiple players gaming, both of which might benefit from efficient multicast schemes, many Internet Service Providers (ISPs) start again to consider solutions based on efficient multicast schemes.

Besides the real-time multicast-based applications, there are some other types of contents, especially for Video-on-Demand, that are popular. These contents are commonly classified as popular contents. The traffic generated by this popular content also accounts for a significant part of the increasing global Internet traffic according to the report from Cisco [3]. As opposed to live Internet TV, this content is fetched frequently by many Internet users, but seldom at the same time. For example, VoD systems allow the subscribers to watch/listen to video or audio at any time they want. In addition, they may offer several versions of video or audio in terms of various QoS in order to support the users under different network conditions. Apparently, the multicast transmission mode is no

longer suitable for disseminating these contents based on the on-demand feature. That is why unicast is still playing the leading role in data transmission. In order to improve on-net content deliveries, an increasing number of Internet operators have started to build a so-called Content Delivery Networks (CDNs). They are a natural extension of distributed storage systems, which normally stores contents with the highest popularity, for instance, web objects (*e.g.* social network), download objects (*e.g.* software updates) and video/audio on servers which are installed at the edge of networks. By reducing the content delivery distance so as to improve transmission latency, these systems indeed help to reduce data traffic in the core network.

Both multicast and distributed storage technologies have been extensively studied. In 2000, the birth of network coding brought fresh ideas and opportunities for improving network performance, also including multicast and distributed storage system. Originally, network coding was a technique that was introduced in the field of coding and information theory, and it was first stated in [5]. Involving algebraic geometry concepts in information theory, network coding is a generic coding scheme which allows information mixing on the intermediate nodes in packet networks at the transport or application layer. The network flow mixing operation is often referred as encoding, and on the other hand, the mechanisms involved in the retrieval of the original information are considered as decoding. These operations fundamentally change the traditional store-and-forward routing scheme, and provide an alternative way of addressing data collision avoidance issue. This change immediately brings its first benefit on multicast performance. Authors in [5] claim that network coding is able to achieve the maximum network capacity for single multicast data communication. In this dissertation, we use the term *Network capacity* for the maximum throughput that a network can achieve for all the potential terminals simultaneously. In addition, we will call *threshold* this theoretical upper-bound on throughput achieved by network coding. Soon after the work by Ahlswede *et al*, it was proved in [65] that linear coding is already sufficient to achieve this threshold. However, network coding might still seem unrealistic, since it requires predetermined centralized and fixed linear coefficients for large networks. The authors in [43] present a distributed random linear network coding approach for multicast network, and they prove that, with a well-chosen size of the fields, randomized codes can provide high success probability for decoding in arbitrary networks. This approach was the first to show the possibility of implementing network coding in practice.

Research in network coding has gained momentum in recent years. Subsequent studies have looked at coding advantage in both wire line and wireless network. The term *Coding Advantage* indicates the benefits, in terms of security, robustness, reliability, and throughput, *etc.*, that we could gain by using network coding mechanisms. For the basic knowledge and the works of the other network coding applications, we refer the reader to Chapter 2 which includes an overview of network coding.

In this dissertation, we define the coding advantage, while referring to throughput gain, by the ratio of network coding throughput to multicast throughput. Although we already know that network coding achieves maximum network throughput, the question of the effective gain to expect from network coding naturally raises. Since 2005, the topic of characterizing throughput gap between network coding and multicast has been extensively studied. The study in [46] initially claimed that the ratio was unbounded theoretically in directed networks. Li *et al* in [67] proved that the coding advantage was no more than 2 in undirected graphs. In 2012, Yin *et al* in [87] showed that there is no throughput gain at all in completely link balanced (or bidirected) networks. The first numerical experiments are addressed in [86]; in this paper, six ISP networks are compared, and none of them presented throughput gain by using network coding. Similar work, in [66], examined random undirected network obtained the same outcome. However, there are some limitations in the evaluation schemes in the former studies, which makes the judgment less convincing. They fail to find efficient algorithms to compute the optimal throughput for *Steiner packing tree* problem. Instead, they either employ approximation algorithms, which is suboptimal, or they choose a brute-force algorithm, to enumerate all the possible multicast trees. That is why they claim that their approach can only evaluate small size networks. In spite of the existence of these limitations, the former research has still given a few hints for further investigations. First, it implies that the throughput gain is very sensitive to the structure of network topologies. Second, since the theoretical results show a drastic contrast to the numerical evaluations, it may imply that butterfly-like instances that display a throughput gap are very rare in real networks.

Although the throughput gain in multicast network remains elusive, the nature of network coding, in addition, alleviates computational complexity of computing maximum multicast flow compared to current multicast routing scheme. Obviously, the full capacity of a network can generally not be fully exploited using

a single multicast tree. This is why some studies, *e.g.*, [13], have investigated the throughput achieved using multiple multicast tree transport protocol. From now on and in the rest of the dissertation, the term *Network Coding* will refer to multicast traffic using of network coding mechanisms. The term *Multicast* will refer to standard multicast mechanisms (mainly packet replication). We call the single multicast tree and multiple multicast tree protocols simply as single multicast and multiple multicast, respectively. Without additional instruction, multiple multicast will always indicate single session. One common method to compute the optimal throughput for a particular routing strategy is to model the corresponding network flow problem in optimization framework. To obtain the maximum throughput and optimal routing scheme for multiple multicast, we need to solve a *Fractional Steiner Tree Packing* problem [47], a well-known *NP-hard* optimization problem. However, the equivalent optimization problem in the case of network coding can be solved in polynomial time. Indeed, to compute the maximum throughput achieved while using network coding, it suffices to run a series of maximum-flow algorithms, one between the server and each client. The maximum-flow problem is a standard problem in graph theory and there exists many efficient algorithms for its resolution [28]. Several papers consider the costs be associated with the use of network coding. For instance, a linear link-cost can model costs associated with bandwidth use or energy consumption. Lun *et al* in [70] have proposed a decentralized approach to compute the minimum-cost multicast subgraphs for network coding. Note that, to achieve the same objective with multicast, one requires a centralized and full knowledge of network topology.

For multicast services, network coding as an advanced technique allows the intermediate nodes to perform calculations. The additional functionalities require software or hardware update at sources and terminals, as well as at intermediate nodes. These changes may disturb other data traffic passing through the network. Moreover, the algebraic operations, such as encoding and decoding, introduce additional workload for network nodes. As a result, they introduce further data processing. These negative effects brought by network coding may be concerns for ISPs to apply this new technique. In view of above-mentioned reasons, some studies have looked at that how to reduce the network coding interference on existing network architectures. Lucani *et al* in [68] aim to limit the volume of network coding flow in order to reduce the computational burden on the intermediate nodes. They propose a hybrid routing protocol as well as its optimization framework, where network coding is only considered as multicast's auxiliary in data transmission. In this problem, the corresponding minimum cost *Steiner tree*

problem for uncoded network flows is still solved by using some suboptimal algorithms as a separate subproblem. In [53] and [55], a *Genetic Algorithm* based on algebraic framework is created to find the minimum possible number of coding nodes achieving a given throughput and which aims to minimize the workload for network update. The problem was claimed to be NP-hard. In this paper, the authors find out that in general, a very small set of coding nodes is already sufficient to provide the maximum throughput benefit.

Network coding not only brings coding advantages in multicast networks, but also in wireless networks, optical networks, and storage systems. The second interest in this dissertation focuses on the network coding application in distributed storage systems. Indeed, in this application, network coding shows a possibility to improve network performance beyond routing considerations. A recent paper [4] studies a revised storage system that stores random linear coded information of the original contents. A coded information packet consists of random linear combinations of all chunks in one generation of an original content. The term *chunk* denotes a fragment of information, which is equivalent to packet in this dissertation. A generation indicates a group of chunks in sequence. When Internet users visit this system to fetch some contents, they will receive some coded information, the amount of which is equivalent or may be a little more than the size of the original content. The users can then retrieve the original content by decoding the corresponding coded information. In [4], it is shown that the revised system maintains very high efficiency in data transmission. The meanings of efficiency are twofold. One denotes that the transmission latency is low, the other indicates that the successful probability of decoding the coded information is high. The survey [20] claims that the distributed storage systems enhance system's reliability and save bandwidth in the inner routing for repairing collapsed storage nodes by using network coding technique in both storage and routing schemes. The authors in [27] point out that the demand of deploying CDNs servers is decreased due to the fact that applying random linear codes reduces the data access blocking probability in CDNs when multiple users want to access one specific information, and as a result, the energy consumption will be reduced. A study in [38] focuses on network coding applications in the storage allocation for wireless video content delivery. It shows that network coded storage system facilitates corresponding mathematical modeling and the corresponding storage scheme provides better performance for file download. The authors in [62, 63, 64] initialize some probability analysis to find the optimal allocation that provides high reliability for Internet users to decode the coded information. However,

it would be interesting to study the mixed storage and routing problems in an optimization framework, since it can be viewed as an extension of conventional transportation problems.

1.2 Motivations and Contributions

Motivated by all these observations, we believe that it is still too early to assert that traditional multicast can be systematically replaced by network coding. The initial theoretical claim of throughput benefit remains relatively elusive, mainly because the multicast throughput maximization problem is difficult to solve. Moreover, local gateways, such as the livebox used by France Télécom, are not often capable of handling highly computational activities, so it must be careful to install encoding and decoding activities on such machines. Therefore, it is essential to revisit the concerning problems, and to weigh carefully the pros and cons brought by network coding. Besides, different storage strategies may have significant impact on related routing behavior, it is interesting to investigate the routing problem in network coding based distributed storage systems.

In this dissertation, we reconsider the problem of evaluating coding advantage in throughput gain within an optimization framework in Chapter 4. The mathematical models and algorithms to maximize multicast throughput we use and propose are sufficient to solve efficiently the problem to optimality in medium size or even larger random network topologies, except those ones with limited number of multicast trees (for those problems, we use simple heuristic algorithms). We confirm that, except in some contrived graphs, the coding advantage in multicast transmission is relatively weak, even in directed graphs which is claimed to have unbounded throughput gain. We perform an exhaustive search in a small network to confirm that the arbitrary networks with throughput gap between network coding and multicast, such as butterfly-like network, are so specific that they almost never occur using random graph generation techniques. Yet, owing to the fact that multicast almost always requires considerable amount of trees to achieve high throughput, network coding will be a rational alternative under the consideration of scalability of network management.

The algebraic operations when doing encoding and decoding bring extra time consuming on nodes, which may delay information delivery in networks. From

the analysis in Chapter 4, we see that using multicast also has some limitations, so we hope to design novel routing protocols, which could reduce negative effects in either network coding or multicast. In Chapter 5, we first consider the problem of minimizing the number of coding nodes and evaluate the trade-off of duplication and coding at nodes. In addition, we propose a hybrid routing protocol, named strategic network coding, to reduce network coding flows by aggregating a limited number of multicast trees. We evaluate all the corresponding problems by solving them using optimization models. Our results indicate that in random networks, only a small set of coding nodes are required for achieving network capacity; the throughput gains of coding increase markedly with graph density; information duplicate onto the outgoing links plays an important role in throughput gain. We reveal that the introduction of a small volume of network coding flow is already beneficial in increasing multicast throughput. However, to achieve high throughput by using strategic network coding, a significant part (about 30%, for the sparsest graphs up to 50% for denser graphs) of the traffic must be processed by network coding, since limited number of multicast trees cannot fully explore the network capacity.

Furthermore, we investigate a transportation problem in network coding based distributed storage systems. In fact, coding scheme frees the importance of single packet, and therefore, allows higher flexibility when storing information in the system and accessing data by clients. Clients only need to receive an amount of coded information of the same size as the original content to be able to decode it. As a consequence, the new content placement policies change routing strategy for content delivery, especially when the accesses on server nodes and/or client nodes become the bottleneck. We extend our optimization model to a general optimization problem, so-called transportation problem with degree constraints, which has not been studied extensively, but that can be widely used in different industrial fields, of course, telecommunication. Finally, we propose a reasonable Lagrangian decomposition approach.

In summary, the main contribution of this dissertation is in providing mathematical models and algorithms for various multicast routing schemes, and clarifying some network coding advantages from numerous numerical experimentations. We present numerical tests and reveal that strategic network coding, a hybrid routing protocol we propose, not only brings significant throughput gain, but also reduces both network coding and multicast weakness. In addition, throughout the studies of network coding application in distributed storage system, we obtain some

preliminary theoretical results for degree constrained transportation problem.

1.3 Outline

The dissertation is organized as follows. In Chapter 2, we provide a brief overview of network coding, multicast network, and distributed storage system, which give fundamental knowledge to understand the rest of the dissertation. We introduce the methodology in accordance with our research, including some reviews of basic graph theory and optimization models in telecommunication.

The main body of this dissertation can be separated into two parts. The first and also the main part embraces Chapters 3, 4 and 5. They focus on providing efficient mathematical models and algorithms for solving maximum throughput problem in different network setting, evaluating coding advantage, and designing new routing strategies, respectively. The other part, Chapter 6, provides an independent preliminary study of routing scheduling problem with network coding based distributed storage systems in wire line networks.

In Chapter 3, we provide efficient algorithms for solving maximum throughput problem when using single multicast tree, in preparation serving for Chapter 4 which evaluates the throughput gain between network coding and different multicast routing settings. In Chapter 4, we revisit the problem concerning coding advantage in terms of throughput gain. The optimization model to solve multiple multicast trees problem is efficient enough to assess medium size (up to 300 links) network. We propose a simple heuristic algorithm to evaluate the throughput gain obtained by limited number of trees. Based on the previous numerical analysis, in Chapter 5, we evaluate the trade-off between the use of coding and duplication at intermediate nodes. We design a novel strategic network coding and introduce mathematical models in optimization framework. In Chapter 6, we study a degree constrained transportation problem, which is suitable for modeling static servers-clients routing scheduling problems in network coding based distributed storage systems.

Finally, in Chapter 7, we summarize our work and suggest some potential extensions to the current work presented in this dissertation.

Chapter 2

Overview of Network Coding

Everything should be made as simple as possible, but not simpler.

-Albert Einstein (1879-1955)

Network coding is a novel step in coding theory where coding can occur within a network. Therefore, it can be viewed as a distributed version of source coding. The technique consists in going beyond simple store-and-forward, and including encoding and decoding functions at the intermediate nodes. It was first introduced and proposed in the year of 2000 in [5]. Research in network coding has gained momentum is recent years.

Subsequent studies have looked at optimal codes, characterization of throughput gains, *etc.* This dissertation does not intend to design network codes for specific network applications, therefore we provide a comprehensive overview of network coding, especially for linear network coding, and random linear coding. The next chapters will not restate network coding mechanism but only apply the flow feature formed by network coding to build our network flow optimization problem. This chapter also includes a few related works embracing wired and wireless network.

2.1 What is Network Coding

In fixed network, on the transmission path from a source to a destination, an intermediate node receives packets (or more generally, *blocks*) on incoming links and forwards them on the outgoing links. Such *store-and-forward* behavior is the norm in most modern networks. With network coding, an intermediate node may instead combine several packets and transmit the resulting encoded packets. The receiver may then retrieve the original information by decoding a sufficient number of encoded packets. It has been shown that, while routing schemes, like unicast, multicast and broadcast, may not always achieve the max-flow capacity of a network, network coding schemes can achieve this capacity [5]. Indeed, network coding increases throughput, for certain scenarios, such as multiple unicast sessions and multicast sessions with the so-called *butterfly* topology shown in Figure 2.1. This figure is an example of a multicast session with a source S and two receivers E and F. The specific structure results in a gain of at least one transmission unit (for a source-destination transmission of two packets) on the link $\overrightarrow{CD}$ where two packets a and b are combined into $a \oplus b$ ($\oplus$ indicates encoding and, in this case, can be implemented by a simple XOR). Note that each receiver is able to decode the packet $a \oplus b$ ane recover bother packets a and b, since it receives, through another path, another packet sufficient for decoding.

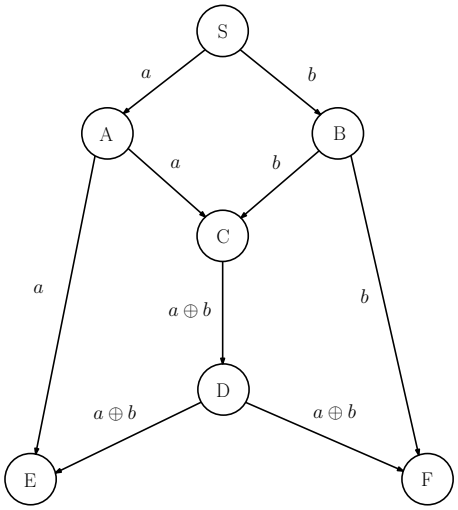


Figure 2.1: Butterfly Network

2.2 Coding and Decoding

The last example shows that the XOR coding scheme is already sufficient to achieve capacity in such a small instance. But in principle, a network code may be any arbitrary mapping, from a set of inputs to some outputs [41], applied at each intermediate node. The goal of network coding is to give a specific set of mappings which allows for the maximum possible transfer of information. It has been shown that simple linear coding is sufficient [65] in order to derive throughput benefit. Koetter and Médard in [59] gave an algebraic formulation of network coding, greatly simplifying both the coding and decoding process. In linear network coding, outgoing packets at a node are linear combinations of incoming packets. The addition and multiplication in these combinations are performed over a finite field, such as $\mathbb{F}_{2^s}$. More details on finite fields will be given in the next section.

Example 2.2.1. *Consider a sequence of incoming packets at a node:* y_1, y_2, y_3. *On the outgoing link, the node may perform linear combinations of the sort:* $x_1 = \alpha_1 y_1 + \alpha_2 y_2 + \alpha_3 y_3, x_2 = \beta_1 y_1 + \beta_2 y_2 + \beta_3 y_3, x_3 = \gamma_1 y_1 + \gamma_2 y_2 + \gamma_3 y_3$, *and transmit the encoded packets* x_1, x_2, x_3.

For combinations of m original packets, the receiver needs at least m encoded packets to decode. More than m packets may be necessary if some combinations are linearly dependent. In selecting linear combinations then, the encoder must make sure they are linearly independent. However, determining what linear combinations each node must perform to ensure linear independence is quite complex. Ho *et al.* in [43] extended this algebraic framework and introduced random linear network coding, where coefficients are chosen at random at each node.

The main advantages of random linear coding are its ease of implementation and its reliability. Random codes allow for a straightforward, distributed routing procedure: At each node where collisions occur, *i.e.*, where the rate of incoming flows exceeds the outgoing capacity, instead of waiting or being dropped, flows could be combined with randomly chosen coefficients. Somewhat surprisingly, this procedure leads, with high probability, to a solution to the problem of information transfer. In [43, Theorem 1], it is shown that the probability of successful decoding process increases exponentially with the coding length l that is the logarithm of the field size $q = 2^l$. Precisely, the probability may be expressed in the form

$1 - \frac{d}{2^l}$, where d is a constant. This feature implies that with a field size large enough, the probability of receiving linearly dependent combinations at terminals is very low.

2.3 A Note on Finite Fields

In general, a field is an algebraic structure over which it is possible to perform the usual arithmetic operations: addition, subtraction, multiplication, and division. In addition to familiar examples, such as the real numbers $\mathbb{R}$ or the complex numbers $\mathbb{C}$, there are also fields which have a finite number of elements. The simplest of these finite fields, also called Galois fields, are realized as integer arithmetic modulo a prime. Indeed, for any prime number p, the ring $\mathbb{Z}/p\mathbb{Z}$ is, in fact, the field with p elements.

For any integer ℓ, it can be shown that there exists a field of size p^ℓ, usually denoted F_{p^ℓ}. When $p = 2$, the finite fields of order 2^m are called binary fields or characteristic-two finite fields. It has been widely used in computer science, since it is suitable to scale values in *bits*. Generally, we apply a so-called polynomial basis representation to construct $\mathbb{F}_{2^m}$, which is constructed as follows:

$$\mathbb{F}_{2^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdot + a_2z^2 + a_1z + a_0 : a_i \in \{0, 1\}\}$$

Then, the arithmetic operations between elements in $\mathbb{F}_{2^m}$ could be treated as polynomial arithmetic, and the arithmetic in coefficients must follow finite field arithmetic operations as well.

Example 2.3.1. *The elements of F_{2^2} are the 4 binary polynomials of degree at most 1:*

$$0 \qquad 1 \qquad z \qquad z+1$$

Addition and multiplication in $\mathbb{F}_{2^2}$ are displayed in Tables 2.1 and 2.2:

At this point, we can return to Example 2.2.1 to see how exactly random linear coding works over finite fields. We could think of y_1, y_2, y_3 as a sequence of symbols that consist of binary vectors of length ℓ. These symbols can also be viewed as chunks of ℓ *bits* as a portion of an original content which is currently transmitted across a network. When three different streams carry, respectively, y_1, y_2, y_3 packets arriving to a certain intermediate node v, we randomly choose

$+$	0	1	z	$z+1$
0	0	1	z	$z+1$
1	1	0	$z+1$	z
z	z	$z+1$	0	1
$z+1$	$z+1$	z	1	0

Table 2.1: Addition in $\mathbb{F}_{2^2}$

$\times$	0	1	z	$z+1$
0	0	0	0	0
1	0	1	z	$z+1$
z	0	z	$z+1$	1
$z+1$	0	$z+1$	1	z

Table 2.2: Multiplication in $\mathbb{F}_{2^2}$

a coding coefficient for each stream. For example, to form the outgoing stream x_1, for every $y_i, i = 1, 2, 3$, we choose $\alpha_1, \alpha_2, \alpha_3$ from a uniform distribution of the elements of F_{2^ℓ}. The linear combination x_1 formed with these random coefficients is then forwarded to every node which receives inputs from node v. It should be noticed that according to [43, Theorem 1], when $\ell = 8$, the successful decoding probability is already very high.

2.4 Cost and Other Concerns

While gains in wired network from network coding seem promising, there are several concerns concerning the use of Network Coding. Theoretical studies on network coding [46, 67] show throughput gains in some specific scenarios, particularly, in multicast sessions and some specific multiple unicast sessions. However, an important element of any gains is the topology of the network. Most studies show gains on the butterfly topology shown in Figure 2.1. More generally, gains can be obtained when there are multiple receivers (either of the same multicast sessions or independent multicast or unicast sessions), several (perhaps independent) paths from source to destination, and sessions that cross paths (have some common links). Generally, a mesh topology would allow such structures. There may be telecommunication operators who can find such topologies in their own networks. It remains to be seen whether all the operators, in general, would find such structures in their networks. On a national level, especially for those countries in Europe, it might be that the network follows more a star or tree topology,

which leaves little opportunity for the structure necessary for network coding gains. On an international level, mesh structures may exist. However agreements on what functions intermediate nodes are allowed to perform on packets must be made. At the receiver, packets need to stored until a sufficient number have been received so as to allow decoding. This implies not only delay in retrieving the original information, but also high buffer requirements. Such buffer requirements are necessary at intermediate nodes as well. In order to avoid wasting transmissions with encoded packets that add linear dependency, intermediate nodes will need to check for linear dependence. This involves checking the rank of the coefficient matrix for each incoming packet. It is important to evaluate whether such extended functionalities in intermediate nodes are negligible against the gain from network coding. There may also be security issues in combining various packets together. In order for buffer sizes and delay to be reasonable, packet combining should be done by generations, where a generation is a sequence of packets. An optimal generation size needs to be determined, and indicated on each packet. Such information, along with information on coefficients used in a given encoded packet, adds to the overhead of the scheme. To the best of our knowledge, this problem has been considered by Sundararajan *et al* who have proposed a rational network coding protocol which is built based on TCP [82]. This special network coding protocol is named TCP/NC. This protocol is later implemented in the OPNET Modeler by Ashutosh *et al* in [61]. In [54] , the authors introduce Network Coded TCP (CTCP) which reduces network coding in a proxy in the network.

In order to confirm whether throughput gain closely relies on network structure, researchers in [66, 86] compared the performance between network coding and conventional routing schemes in networks of real operators as well as random generated graphs. They found out that the throughput gain is rare on such random topologies. But instead of throughput gain, they pointed out that network coding does help in reducing computational complexity for bandwidth allocation problem and also on reducing the burden on network flow control. However, as we have already discussed in Chapter 1, some limits on the methods applied in these references make the question on throughput gain still elusive, and leave us space for some improvements.

Concerning to the additional computational complexities coming along with encoding and decoding, there have been significant efforts made to tailor and adjust network coding. Authors in [53] and [55] provide an algebraic framework to min-

imize the usage of coding nodes and achieve the same throughput. Lucani *et al* in [68] suggest using network coding only as an auxiliary tool in some particular network circumstance where coding is not easy to be performed, for instance, in optical networks. Applying network coding over application-layer overlay multicast would be another remedial solution to eliminate the engineering concerns on coding and decoding. Reference [66] first proposed this idea and provided an information flow formulation for it. In overlay networks, coding and decoding are only implemented in terminals, the communication between terminals, as usual, is in accordance with End-to-End principle, so that we also protect the initial network control philosophy. However, we deem that this method introduces more cost and delay because a conversion from physical layer into application layer and then back to physical layer in each terminal is necessary.

2.5 Other Applications

Although there has been some concerns in applying network coding, with some appropriate adjustments, gains from network coding seem still promising for data routing in wired network. Besides, network coding evolves fast in many other network implementations and applications, both in wireline and wireless networks.

There are several scenarios that we broadly categorize as application-layer schemes where network coding might provide gains. One such example is content distribution. In peer-to-peer (p2p) content distribution, the last rare block usually causes delay in file transfer completion. With network coding, where blocks are encoded together into many combinations, such rare blocks no longer exist. Avalanche [36] is a scheme for content distribution of large files. Each node in the network generates and transmits encoded blocks of the original file. The randomization introduced by network coding in this case, has many benefits. Each node does not need to keep track of which block it must send to which neighbor. Since each node encodes before transmitting, the resulting combining packet will, with a high probability, aid in decoding other blocks. Furthermore, the inherent redundancy of information due to this scheme helps when nodes accidentally leave the system - that is, node crashes or departures do not imply that rarest blocks are missing. Microsoft has made claims of its use in software downloads, called Microsoft Secure Content Distribution (MSCD).

Distributed storage system is another example where network coding might im-

prove performance. System reliability is one of the main issues addressed by researchers and engineers in this application. Introducing redundancy and storing them across different storage is a common way to enhance system reliability. Generally, the redundancy consists of either copies of original content or coded information which is generated by using erasure coding technique. As well as reliability improvement, studies in [4, 20] discovered new benefits in improving system utilization and maintenance when using coded information and intra network coding based routing in the system. Simulation results in [4] showed that a random linear coded distributed storage system not only preserves the same level of reliability as erasure codes do, but unlike erasure codes, which require to store the whole coded information in the system, random linear coded information could also be generated *on the fly*. As a natural extension of linear network coding, coded information could also be recombined whenever necessary. As is demonstrated in [20], such freedom is helpful in reducing bandwidth consumption when a system needs to recover some failed storage nodes.

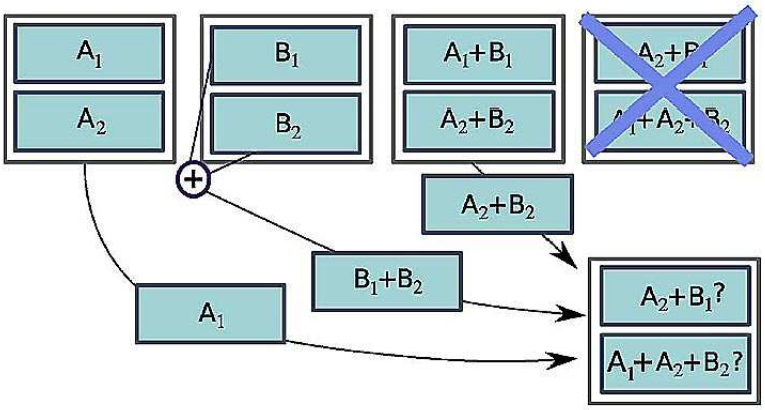


Figure 2.2: The graph is from [20]. It demonstrates a storage system applying network coding to recover a failed node. The example use a (2,4) Erasure Code to store the original content and to generate the redundancy, but we can use random linear codes instead, and the repairing process still functions.

Beyond wired network applications, network coding may minimize energy consumption and also reduce transmission delay in a wireless scenario, through fewer transmissions. The wireless scenario naturally provides opportunities for network coding. The wireless link is not a point-to-point link (even though it is treated as such in current implementations); the broadcast nature of wireless transmission can be exploited. The broadcast nature in fact creates information redundancy in the environment, where many nodes may listen to transmissions, even those not destined for them. A transmitting node can then combine packets known to be in

the "environment" and thus decodable by the receiver. Several protocols which take advantage of network coding have been proposed in the literature [11, 49, 50]. Among these protocols, reference [51] showed an intuitive example on the possible network coding gain in wireless networks. As it is shown in Figure 2.3, in this simple network with three nodes, there are two flows, one from node C to node B and another in the opposite direction. Without network coding, 4 time units are need to transmit 4 packets. With network coding, once the center node A has packets x_1 and x_2, it only needs to transmit (broadcast) one packet, $x_1 \oplus x_2$. Since node B has packet x_2 (it transmitted that packet), it can easily decode x_1 from $x_1 \oplus x_2$. Similarly, node A is able to recover x_2. The same information is thus transmitted in 3 time units instead of 4.

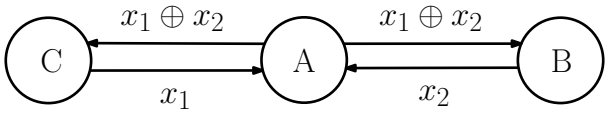


Figure 2.3: A wireless example

There are, however, several practical issues in the wireless scenario as well. A wireless link undergoes fading and thus the transmission rate is varying. A coded packet may need to be transmitted at a lower rate if the link to one of its receivers suffers from fading. In the example of Figure 2, if the link between nodes A and B is fading and thus has a lower rate than the link between nodes A and C, then the transmission of the coded packet will be done on a lower rate. This implies not only smaller throughput gains, but perhaps increased error probabilities owing to the weaker link. In reality, wireless networks are more complex than this example, and there is interference from other transmissions in the area. Interference may cause bad reception at some receivers, and so needs to be taken into account during the scheduling and coding functions. Regarding practical application, Kim *et al* [56] analyze the performance of TCP/NC in lossy wireless networks where they show a significant throughput gain.

2.6 Summary

Network coding indeed has huge potential benefit in extensive variety of network implementations and applications. But single network coding solution satisfying all networking problems does not exist. Network coding must be tailored and adjusted with regard to the type of network, the network application demand and all the other related concerns for specific use. On the other hand, The use

of network coding must consider the trade-off of benefit and the interference of existing network architecture. This dissertation aims to clarify the practical benefits of network coding and to provide some guidance in applying and designing network coding routing systems that improve multicast service performance in wired network.

Chapter 3

Algorithms for Finding Unsplittable End-to-End Throughput in Multicast Network

The most incomprehensible thing about the universe is its comprehensibility.

-Albert Einstein (1879-1955)

3.1 Summary

Multicast protocols are very efficient mechanisms for sending a same content simultaneously to several users in a telecommunication network. A multicast distribution tree is featured in spanning some given terminals (if we consider the source as a special terminal). Finding such a distribution tree in a network is similar to find a Steiner tree in a graph which finds a minimum cost tree spanning some terminals [37]. If one considers maximizing the throughput in such a tree, the related Steiner tree problem is called a bottleneck problem and becomes polynomial. The algorithms running in polynomial time for *Bottleneck Steiner Tree* (BST) problem have also been widely studied. The algorithms mainly rely on the modified version of the algorithms for solving the *Maximum Spanning tree* problem. As a natural extension of BST, we consider the *Full Bottleneck*

Steiner Tree (FBST) problem in which only a subset of terminals are required to be the leaves of the multicast tree, and we create two algorithms for solving this special problem. The algorithms apply modified versions of Dijkstra's algorithm and Kruskal's algorithm as their core sub-algorithms. The running time of the first one is $O(|S|^2|T|)$, where $|S|$ and $|T|$ denote the number of Steiner nodes and terminals, respectively. Then we provide an alternative which improves the complexity to $O(|E| + |E| \log |E|)$ where $|E|$ represents the number of edges in graphs. This algorithm only makes use of Kruskal's algorithm as a core algorithm which can be easily implemented.

3.2 Introduction

When the cost of expanding network capacity is considerably high, the routing schemes for data transmission may need to be revised in order to adapt to the increasing data traffic. The performance of a given routing strategy can be evaluated from several aspects, including the measurement of transmission delay, network throughput or network reliability, security and robustness. In this dissertation, we mainly focus on measuring network throughput gain. Network throughput considers the amount of data traffic which a communication link or network access can carry in bits per second. The maximum throughput on a specific communication link can be seen as the capacity of the given link. Under the end-to-end principle, the amount of data traffic which a communication path or a multicast tree can carry is denoted by so-called end-to-end network throughput. It is often restricted by the minimum link capacity on the transmission routes.

In this context, the routing strategies have significant impact on the throughput achievement. Routing strategies are the set of mechanisms that allow to decide how data is transfered in telecommunication networks. Most of the traffic is still point-to-point, i.e., between two clients, terminals or routers. However, with the development of Web TV, video-conferences, distant learning and other live applications, the share of point-to-multipoint or multicast traffic [85], is becoming more and more significant. This is the reason why multicast routing problems have received much attention and been extensively studied. By replicating data over several output links, multicast routing protocols allow to transfer the same content simultaneous to several destinations and, as a result, avoids unnecessary flow replication and save considerable amounts of bandwidth [60].

Designing "optimal" multicast systems often requires to consider Steiner tree problems, a special case of the spanning tree problem. Given an undirected graph $G = (V, E)$, a spanning tree is an acyclic subgraph spanning (containing) all nodes in V. If there are some weight value w_e associated to the edges $e \in E$, a minimum spanning tree, is a spanning tree of minimum weight. This problem is one of the easiest graph problems. If the graph is only required to span a subset of nodes, usually called terminals ($T \subseteq V$), then the problem becomes the Steiner tree problem and the nodes that are not terminals (i.e., in $V \setminus T$) are called Steiner nodes. This problem has been largely studied (see for instance [**?**]) and is known to be NP-hard in most cases. Several approximation algorithms have been proposed, and G. Robin shows that, so far, the best performance ratio of Steiner tree problem in graphs is 1.55 in [78]. In these problems, the objective consists in minimizing the sum of edge weights and can thus be described as *min-sum* problems. While designing a multicast tree, one important objective is to use as little resources as possible. Hence, minimizing the number of edges, or the sum of edge costs, required to span all servers and clients is a relevant Steiner tree problem.

A metric often related to Quality of Service (QoS) is the ratio of flow over capacity, congestion occurring when this ratio is close to 1. As a consequence, it is also relevant to consider problems in which the capacity available in a tree for a multicast communication is as large as possible (given a set of available resources). In this case, it is not the sum of weights or costs that should be minimized, but rather the minimum edge capacity (to maximize). These problems can be seen as *min-max* (or *max-min*) variants of the Steiner tree problem, also often known as bottleneck problems. Although regular Steiner tree problems are difficult (most of the time NP-hard), the bottleneck counterparts are usually polynomial problems. An algorithm in $O(|E|)$ is proposed in [24] to solve the Bottleneck Steiner Tree (BST) problem in its min-max version. Similarly, a linear time algorithm for computing optimal bottleneck multicast tree in directed graphs is proposed in [35]. A special case of BST, known as Full Bottleneck Steiner Tree (FBST) problem, consists in adding the requirement that, in the resulting tree, each terminal node should be of degree one (leaf-nodes). An algorithm also in $O(|E| \log |E|)$ is proposed in [15] to solve this problem. This algorithm is based on the assumption that the input graph is complete and that the edge weights in the subgraph induced by the Steiner nodes should be totally ordered (no two edges of the same weight).

In this chapter, we summarize the bottleneck network flow problems, then we investigate the bottleneck Steiner tree problems in relation with the throughput maximization problem in multicast trees. The main contribution in this chapter is that we provide two polynomial time algorithms on the full bottleneck Steiner tree problem. These algorithms can be easily implemented, since they all make use of the fundamental algorithms that solve simple path or tree network flow problems as their core sub-algorithm.

The rest of this chapter is organized as follows. In Section 3.3, we summarize some network flow problems and their bottleneck variants. We clarify the relationship between the problems and the possible applications in multicast network. We provide some important proof of the lemmas missing in the literature. In Section 3.4, some preliminary results concerning bottleneck problems are recalled. In Section 3.5, an $O(|S|^2|T|)$ algorithm for FBST problem is proposed and in Section 3.6, a less complex algorithm for FBST is proposed. Finally, we give some conclusions in Section 3.8.

3.3 Bottleneck Network Flow Problems

The problems of finding the end-to-end throughput in a given network are often referred to as bottleneck versions of network flow problems. These problems are often classified as combinatorial optimizations. For example, the *bottleneck shortest path* problem [48] applies a modified version of *Dijkstra shortest path* algorithm and finds a single widest path in a capacitated graph. Problems of this kind are suitable, for instance, in the unsplittable unicast routing management. Unsplittable unicast allows using single path to carry network flow. The *maximum flow* problem [28] involves looking at the maximum flow between a source and a terminal when multiple paths are allowed. The problem is first introduced as a military application, but then engineers applied it in telecommunication networks to find the maximum flow rate that can be sent from a source to a particular terminal. Besides the single path and multiple paths scenarios, there are also a generalization of these problems called *k-splittable flow problem* [9] which help to find the best unicast routings with bounded paths. For multicast protocol, there are also corresponding tree-based network flow problems which provide methods for allocating single multicast tree [35], multiple trees [47], as well as a limited number of trees. Some of the above-mentioned problems are easy to be solved in polynomial time, but the others are NP-hard problems. Normally, there are two

common way for solving these problems: applying mathematical programming skills, especially linear programming or integer programming to solve the corresponding mathematical models; or designing algorithms based on graph theory. Selecting an approach to solve one problem depends on the problem itself. There are abundant efficient state-of-the-art algorithms in graph theory that solve fundamental network flow problems, for instance, *shortest path* problem, and *maximum/minimum spanning tree* problem. For those $NP-hard$ problems, it is popular to apply algorithms that proved good approximations or suboptimal solutions instead of the optimal solution to reduce computational complexity. It should be noted that, for different types of graphs, such as directed, undirected, or bidirected graphs (the definition of the graphs is given in Section 3.3.1), the methods for solving a particular problem may vary.

In the following, we introduce the network flow problems and corresponding algorithms which are related to our research in this chapter. In addition, we define the bottleneck problems which can be applied to the corresponding telecommunication problem that finds the maximum throughput when single multicast tree is used.

3.3.1 Notations

The graphs are classified according to whether the edges have orientation. $G = (V, E, C)$ denotes an undirected graph, which comprise a set of nodes V and a set of edges $E = \{\{i, j\}|i, j \in V\}$ in which $\{i, j\}$ is an unordered pair of nodes with respect to a particular edge. $D = (V, A, C)$ denotes an directed graph, where A represents a set of arcs $A = \{(i, j)|i, j \in V\}$ in which (i, j) consists of ordered pairs of nodes. It should be noted that, in graph theory, an oriented edge is usually called an arc. The bidirected graphs $B = (V, A, C)$ are special cases of directed graph, in which there are two arcs in opposite directions between a pair of nodes. Sometimes, we index the edges and arcs, and denote them by $e \in E$, $a \in A$ in this dissertation. The attribute $\mathcal{W}$ indicates a mapping which associates a value $w_e \geq 0$ to each edge $e \in E$ or $w_a \geq 0$ to each arc $a \in A$. According to the context, these values will be either considered as weights or as capacities. In the set V, some nodes are considered as terminals which form a subset $T \subseteq V$. The rest of the nodes that act as the intermediate nodes in telecommunication networks are named Steiner nodes $S = V \setminus T$.

In undirected graphs, given a set of edges $F \subseteq E$, we denote $V(F)$ the set of

nodes induced by F. We denote $V(v)$ the set of neighboors of v, that is the set of nodes linked to v by an edge of E. As a result, $S(v)$ indicates the set of Steiner nodes which link to node v. A similar definition also applies to $T(v)$. We denote by $E(v)$ the set of edges with one ending node at v. Especially, we name the set of edges that have only one end at nodes $t \in T$ the terminal-Steiner edges and denote them by $E(T)$. Assume that some edges are subtracted from some edges from G, there are m disconnected components in a set of subgraph, they are denoted by $G_i, i = 1, 2, \ldots, m$. It is obvious that these components satisfy the simple conditions as follows:

$$V_i \cap V_j = \emptyset, E_i \cap E_j = \emptyset \quad \text{if } G_i = (V_i, E_i), G_j = (V_j, E_j) \text{ and } i \neq j, i, j \in \{1, 2, \ldots m\}.$$

Moreover, for any subset of nodes $W \subset V$, we denote $\delta^-(W)$ and $\delta^+(W)$ the set of arcs leaving and entering W in the graph G, respectively. For a given subset of edges $F \subset E$, we will sometimes use the compact notation $w(F)$ instead of $\sum_{e \in F} w_e$. We denote by $d_G(v)$, or simply $d(v)$, the degree (number of adjacent nodes) of node v in the graph G. Similarly, $d_F(v)$ will denote the degree of v in a subgraph F of G.

We also use the standard graph notations for cuts in graphs: if S is a non-empty subset of V, then $\delta(S)$ denotes the set of edges between S and $\overline{S} = V \setminus S$. For two subsets of vertices S_1 and S_2 (not necessarily disjoint), we will also denote $E(S_1, S_2)$ the set of edges between the two subsets:

$$E(S_1, S_2) = \{uv \in E : \ u \in S_1, v \in S_2\}.$$

Hence $\delta(S) = E(S, \overline{S})$. We denote $E(S)$, instead of $E(S, S)$, the set of edges between nodes of S and by $G(S)$ the subgraph of G induced by the set of nodes S, i.e., $G(S) = (S, E(S))$.

3.3.2 Path Problems

Shortest Path Problem

In a directed graph D, a path is a walk traversing a set of nodes following the directions of corresponding arcs and without any repetition of nodes. A $s-t$ path is a path between the node s (the source) and t (the terminal). We denote $\mathcal{P}(s, t)$ the set of all possible paths between s and t. The shortest $s - t$ path problem is to find a $s - t$ path in which the sum of the edges' weights is minimized. This is

a classical combinatorial problem which has been widely studied [6]. The $s-t$ shortest path problem has the following formulation:

$$\text{minimize} \quad \sum_{(i,j)\in A} w_{(i,j)} x_{(i,j)} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{\{j:(i,j)\in A\}} x_{(i,j)} - \sum_{\{j:(j,i)\in A\}} x_{(j,i)} = b_i \qquad i \in V, \tag{3.2}$$

$$b_s = 1, b_t = -1, \tag{3.3}$$

$$b_i = 0, i \neq s, t \tag{3.4}$$

$$x_{(i,j)} \in \{0,1\}, \qquad \forall (i,j) \in A \tag{3.5}$$

where the binary variables $x_{(i,j)}, (i,j) \in A$ in constraints (3.5) indicate whether the arcs are being used. When the constraints (3.5) are relaxed, the constraint sets of the problem can be transformed into the canonical form of linear programming $\mathcal{A}x = b$, where b is a column vector of integer values that correspond to the right-hand side in constraint sets (3.2), and the matrix $\mathcal{A}$ is composed of 1,0,-1 corresponding to the coefficients of the variables in (3.2). It is stated in [14] that in the minimum cost network flow problem (MCNF), the matrix $\mathcal{A}$ in this problem is Totally Unimodular (TU), in which each subdeterminant of $\mathcal{A}$ is 0, +1, or 1. The property of TU is given in Theorem 3.3.1. According to Theorem 3.3.2 and Corollary 3.3.1, we can find an integer optimal solution, when we solve the LP relaxation formulation, hence the solution is also the optimal solution of the original integer programming problem. Hence, the shortest path problem is polynomial time solvable.

Theorem 3.3.1 ([81]). *A matrix $\mathcal{A}$ is totally unimodular if and only if each collection R of rows of $\mathcal{A}$ can be partitioned into classes R_1 and R_2 such that the sum of the rows in R_1, minus the sum of the rows in R_2, is a vector with entries $0, \pm 1$ only.*

Theorem 3.3.2 ([80]). *Let $\mathcal{A}$ be a totally unimodular $m \times n$ matrix and let $b \in \mathbb{Z}^m$. Then the polyhedron*

$$P := \{x \mid \mathcal{A}x \leq b\} \tag{3.6}$$

is integer.

Corollary 3.3.1 ([80]). *Let $\mathcal{A}$ be a totally unimodular matrix and let b, c be integral vectors. Then both problems in the LP-duality equality*

$$\max\{c^T x : \mathcal{A}x \leq b\} = \min\{y^T b : y \geq 0 and y\mathcal{A} = c\} \tag{3.7}$$

ave integral optimal solutions.

Many algorithms have been proposed to solve the shortest path problem. Dijkstra's algorithm is the most famous one that solves the problems in graphs associated with non-negative weights. It often arises as a sub-algorithm of the algorithms for solving other network problems. In this chapter, we apply a modified version of Dijkstra's algorithm to find the bottleneck multicast tree and use it as a sub-algorithm in full bottleneck Steiner tree problems.

Dijkstra's algorithm is a greedy process, which, in fact, can find a sequence of shortest paths between a single source and all the other nodes in the graph. However, it may take longer time to find all $s-t$ paths where $t \in T$ than only one $s-t$ path required by the original shortest path problem, since the algorithm has to traverse all nodes in the graph. On the contrary, finding a specific $s-t$ shortest path will terminate the algorithm once the terminal t is reached. The construction of the shortest path between s and a particular node can be rebuilt from the information given in the so-called *distance labels* that are being updated during the implementation of the algorithm. A distance label is a $2-tuple$ containing two attributes and it is assigned to each node. The first element in the tuple, denoted by $\phi(v)$, indicates the optimal length of the path between s and node v. It should be noticed that the distance label of one specific node is not global optimal until the algorithm stops. In order to construct the paths according to the optimal labels, the second element in the tuple, denoted by $\rho(v)$, records the predecessor of node v, where the path comes from. Since the paths never form cycles, Dijkstra's algorithm indeed explores a shortest path spanning tree which is rooted at s in a directed graph.

In undirected graph, we can apply Dijkstra's algorithm on a complete bidirected graph that is converted from the original undirected one. A complete bidirected graph is a bidirected graph with the same weights on the arcs in opposite directions.

Assume that Q contains a set of nodes, and for a given node v, the distance label $(\phi(v), \rho(s))$. The version of Dijkstra's Algorithm in [77] is shown in Algorithm 3.3.2.1.

The running time of the algorithm is $O(|V|^2)$. Fredman and Tarjan in [31] apply a Fibonacci heap and improve the efficiency of the algorithm performance, which

Algorithm 3.3.2.1 : Dijkstra's Algorithm

Input: $Q := \{s\}$; $\phi(s) = 0$; $\rho(s) = s$; $\phi(i) = \infty$; $\rho(i) := null$; $\forall i \in V, i \neq s$;
Output: A shortest paths spanning tree;

```
1. while Q ≠ ∅ do
2.    i = arg min_{j∈Q} φ(j); Q = Q \ {i};
3.    for each outgoing arc a_(i,j) do
4.       if φ(j) > φ(i) + w_(i,j) then
5.          φ(j) := φ(i) + w_(i,j); ρ(j) := i;
6.          add j to Q if it does not already belong to Q;
7.       end if
8.    end for
9. end while
```

reduces the running time to $O(|A| + |V| \log |V|)$, where $|A|$ represents the number of edges, and $|V|$ denotes the number of nodes.

Bottleneck Path Problem

The $s - t$ bottleneck path problem aims to search a path between s and t in which the minimum/maximum weight of the edges or arcs is being maximized/minimized. In undirected graphs, two bottleneck path problems can be defined using either min-max or max-min criteria:

$$\mathrm{BP}^{\downarrow\uparrow}(s,t) : \min_{p \in \mathcal{P}(s,t)} \max_{e \in p} w_e \tag{3.8}$$

$$\mathrm{BP}^{\uparrow\downarrow}(s,t) : \max_{p \in \mathcal{P}(s,t)} \min_{e \in p} w_e. \tag{3.9}$$

It should be noticed that, without additional motification, the formulations for defining problems in undirected and directed graphs can be converted easily by replacing e/a to a/e. The shortest path problem and bottleneck path problem are highly related, but the bottleneck path problems are easier to be solved. For the undirected graph scenarios, the authors in [48] give a linear time $O(|E|)$ algorithm to solve bottleneck path problems (3.8). For the directed graph scenarios, a modified version of Dijkstra's algorithm is sufficient to find the bottleneck paths between s and all the other nodes in a given graph. The modified algorithm is given in [35], it only changes initializations of the distance labels, and the way for updating the distance labels. Algorithm 3.3.2.2 still runs at $O(|V|^2)$. The authors of [35] propose an improved implementation of the modified Dijkstra's algorithm using the presorted weights. They prove that the bottleneck path problem is linear time solvable; the running time of the updated algorithm is $O(T(m))$, where $T(m)$ is the time complexity of the algorithm used to sort the

Algorithm 3.3.2.2 : Modified Dijkstra's Algorithm

Input: $Q := \{s\}$; $\phi(s) = \infty$; $\rho(s) = s$; $\phi(i) = 0$; $\rho(i) := null$; $\forall i \in V, i \neq s$;
Output: A shortest paths spanning tree;

```
1. while Q ≠ ∅ do
2.    i = arg min_{j∈Q} φ(j); Q = Q \ {i};
3.    for each outgoing arc a_(i,j) do
4.       if φ(j) < max{φ(i), w_(i,j)} then
5.          φ(j) := max{φ(i), w_(i,j)}; ρ(j) := i;
6.          add j to Q if it does not already belong to Q;
7.       end if
8.    end for
9. end while
```

weighted arcs.

3.3.3 Spanning tree problems

Minimum/Maximum Spanning Tree Problem

A tree is an acyclic connected subgraph. Given a subset of nodes $S \subseteq V$, a tree is said to span S if the tree contains all nodes of S (each node of S is adjacent to at least one edge of the tree). We denote by $\mathcal{T}(S)$ the set of trees spanning S. When $S = V$, a tree in $\mathcal{T}(V)$, or $\mathcal{T}$ for short, is simply called a spanning tree. In the minimum spanning tree problem, the total weight of the spanning tree must be minimized:

$$\mathrm{MT}^{\downarrow} : \min_{t \in \mathcal{T}} \operatorname{sum}_{e \in t} w_e. \tag{3.10}$$

The minimum/maximum spanning tree problem in a directed graph has to be redefined, and it will be defined in the section describing the bottleneck variants. There are two algorithms that are usually quoted and used for solving the minimum and maximum spanning tree problem, Prim's algorithm and Kruskal's algorithm, respectively. Since the algorithm which we create to solve the full bottleneck Steiner tree problem, is inspired and built upon Kruskal's algorithm, we focus on the detail of Kruskal's algorithm in this section. This algorithm starts by sorting the weights of edges in a non-decreasing order. Then the edges are picked up in the non-decreasing order until all the nodes are connected. In order to avoid cycles in the final subgraph, an edge will not be chosen when it causes cycles.

Assume that there are $|E| = m$ edges, and after sorting the edges, the weights of which are denoted by $w_{e_1} \leq w_{e_2} \leq \ldots w_{e_m}$. In the algorithm, we use the

set E_p to keep track of the edges being picked up during the implementation of the algorithm, and it is initially set to $\emptyset$. Kruskal's algorithm [18] is shown in Algorithm 3.3.3.1.

Algorithm 3.3.3.1 : Kruskal's Algorithm

Input: $k := 1; E_p = \emptyset$;
Output: A minimum spanning tree;

```
1. while G = (V, E_p) is not a connected graph do
2.    let e_k = e_(i,j);
3.    if i and j is not connected in G = (V, E_p) then
4.       add e_(i,j) to E_p;
5.    end if
6.    k := k + 1
7. end while
```

The running time of the algorithm is $O(|E| \log |V|)$ with simple data structure [81]. It should be noticed that the algorithm is also suitable for negative weight graphs, so it is easy to adapt the algorithms solving $\mathrm{MT}^{\downarrow}$ to the maximum weight spanning tree problem, which is defined as follows:

$$\mathrm{MT}^{\uparrow} : \max_{t \in \mathcal{T}} \sum_{e \in t} w_e. \tag{3.11}$$

We denote respectively by $\mathcal{T}^{\uparrow}$ and $\mathcal{T}^{\downarrow}$ the set of minimum and maximum spanning trees in G.

Bottleneck Spanning Tree Problems

The min-max and max-min counterparts of these spanning tree problems define two families of bottleneck problems in undirected graphs:

$$\mathrm{BT}^{\downarrow\uparrow} : \min_{t \in \mathcal{T}} \max_{e \in t} w_e, \tag{3.12}$$

$$\mathrm{BT}^{\uparrow\downarrow} : \max_{t \in \mathcal{T}} \min_{e \in t} w_e. \tag{3.13}$$

We denote respectively by $\mathcal{T}^{\downarrow\uparrow}$ and $\mathcal{T}^{\uparrow\downarrow}$ the corresponding sets of bottleneck spanning tree with min-max and max-min criteria.

Solving the bottleneck spanning tree problem in undirected graphs is not harder than finding a minimum/maximum spanning tree according to Lemma 3.3.1. Applying Prim's algorithm or Kruskal's algorithm not only helps to find the

minimum/maximum spanning tree, but also solves the problem of the bottleneck version.

Lemma 3.3.1. *$\mathcal{T}^{\downarrow} \subseteq \mathcal{T}^{\downarrow\uparrow}$ and $\mathcal{T}^{\uparrow} \subseteq \mathcal{T}^{\uparrow\downarrow}$.*

Proof. Both results are equivalent. We prove the first one: suppose that $t_1 \in \mathcal{T}^{\downarrow}, t_1 \notin \mathcal{T}^{\downarrow\uparrow}$. Denote w_1 the maximum edge weight in t_1 and $w^{\downarrow\uparrow}$ the maximum edge weight in any bottleneck tree. By assumption, we have that $w_1 > w^{\downarrow\uparrow}$. Consider an edge $e_1 \in t_1$ of weight w_1 and an edge $e_2 \notin t_1$ of weight $w^{\downarrow\uparrow}$. Then the tree $t_1 \setminus \{e_1\} \cup \{e_2\}$ is a spanning tree with a total weight strictly lower than $w(t_1)$. It contradicts the fact that $t_1 \in \mathcal{T}^{\downarrow}$, hence $\mathcal{T}^{\downarrow} \subseteq \mathcal{T}^{\downarrow\uparrow}$. □

As an immediate consequence, any path p^*_{st} in a maximum spanning tree is also a bottleneck path in $BP^{\uparrow\downarrow}$.

In directed graphs, spanning trees have to be redefined. The directed spanning tree includes a single root, and some directed paths originated at the root and ended at the other nodes in the graph. These two properties guarantee the bottleneck tree we found can be used as a multicast tree when the directed graph is considered as a model for a telecommunication network. As a result, the max-min bottleneck spanning tree can be used to evaluate the best end-to-end throughput when a source multicast information is sent to all nodes in the network (except itself) and when the network flow is unsplittable. An unsplittable multicast allows single multicast tree to carry network flow.

Solving the bottleneck spanning tree problem is as easy as minimum spanning tree. We will now illustrate how to solve this problem in its max-min version. The same argument can be easily adapted to the min-max version. In the bottleneck spanning tree for a given directed graph, the paths which include the bottleneck arc(s) are the bottleneck paths between the source and the nodes behind the bottleneck on those paths. And the bottlenecks between the source and the other nodes that lie on the paths exclusive of the bottleneck arc(s) are larger than the weight of the bottleneck arc(s). It implies that finding a sequence of bottleneck paths could also acquire a bottleneck spanning tree with the same optimal bottleneck value, though the tree may not always be the same as the one found by other algorithms. Hence, the bottleneck spanning tree problem in directed graph can be solved by using existing Algorithm 3.3.2.2 that solves the bottleneck path problem.

3.3.4 Steiner Tree Problems

Minimum Steiner Tree Problem

A Steiner tree is simply a minimum weight tree spanning a subset of nodes $T \subseteq V$ and $T = \{s, t_k | k = 1, 2, \ldots, k\}$ (called terminals). The problem to find the optimal Steiner tree is called Steiner tree problem. It has been widely studied. Different versions of Steiner tree problem has emerged. In this dissertation, we focus on the Steiner tree problem in graphs, since it is a problem that is highly related to our multicast network applications. We denote the objective of this problem as follows:

$$\mathrm{ST}^{\downarrow} : \min_{t \in \mathcal{T}(T)} \operatorname{sum}_{e \in t} w_e. \tag{3.14}$$

We assume that there is one special terminal, called source, such that on one tree, there is always a path between the source s and each terminal. But the paths between source s and the other terminals are not independent. They usually share some common arcs. One set of path constraints in (3.2) is used to guarantee the connectivity between a source and one terminal through a path. There are $k = |T| - 1$ such path constraints promising the connectivities among terminals throughout several (s, t_k) paths. In the literature, they are often called degree conservation constraints or flow conservation constraints. But, in our case, they are dummy or virtual flows, since these constraints are only used to indicate the connectivities among nodes. The dummy flow variable on each arc is denoted by r_a^k. The relationship between the dummy flows and the real usage of the arcs to build the Steiner tree is given in the set of constraints (3.16). The binary variables $x_a, a \in A$ are the arcs' indicators. When $x_a = 1$, it denotes that the corresponding arc is being chosen. One possible mathematical formulations for the Steiner tree problem is given [37] as follows:

$$\text{minimize} \quad \sum_{a \in A} w_a x_a, \tag{3.15}$$

$$\text{s.t.} \quad x_a \geq r_a^k, \qquad \forall a \in A, k = t_k, \tag{3.16}$$

$$\sum_{a \in \delta^-(v)} r_a^k - \sum_{a \in \delta^+(v)} r_a^k = b_v^k(1), \qquad \forall v \in V, k = t_k, \tag{3.17}$$

$$r_a^k \geq 0, x_a \in \{0, 1\} \qquad \forall a \in A, k = t_k \tag{3.18}$$

where the constants b_v^k are defined as:

$$b_v^k(x) = \begin{cases} -x & \text{if } v = s \\ x & \text{if } v = t_k \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

It should be noticed that this formulation is based on directed graphs, but it can be easily adapted to undirected and bidirected instances.

The Steiner tree problem is known to be NP-hard in most cases. But if $|T| = 2$, the problem reduces to shortest path problem, and if $|T| = |V|$, the problem becomes the minimum spanning tree problem. These are the only two known scenarios that can be solved in polynomial time. Different formulations have emerged and a comprehensive survey on this problem can be found in [37]. Moreover, as we mention in Section 3.2, several approximation algorithms have been proposed. The problem in which the total weight of the tree is maximized, which has less meaning in real problem applications, has been much less studied:

$$\mathrm{ST}^{\uparrow} : \max_{t \in \mathcal{T}(T)} \sum_{e \in t} w_e \tag{3.20}$$

The set of minimum and maximum Steiner tree spanning T are denoted by $\mathcal{T}^{\downarrow}(T)$ and $\mathcal{T}^{\uparrow}(T)$, respectively.

Bottleneck Steiner Tree Problem

We consider the alternatives of the Steiner tree problem obtained by changing the min-sum and max-sum objectives into min-max and max-min, then we obtain the two variants of Steiner bottleneck problems:

$$\mathrm{BST}^{\downarrow\uparrow} : \min_{t \in \mathcal{T}(T)} \max_{e \in t} w_e \tag{3.21}$$

$$\mathrm{BST}^{\uparrow\downarrow} : \max_{t \in \mathcal{T}(T)} \min_{e \in t} w_e \tag{3.22}$$

We call $\mathcal{T}^{\downarrow\uparrow}(T)$ and $\mathcal{T}^{\uparrow\downarrow}(T)$ the set of min-max and max-min bottleneck Steiner trees, respectively.

When we restrict our attention to the max-min problem 3.22, and in addition, we assume that the weights in graphs are considered as the capacities in telecommunication network, solving this problem is equivalent to find the end-to-end

throughput in single multicast tree protocol. Compared to the bottleneck spanning tree problem, the bottleneck Steiner tree problem is more rational in modeling multicast service, since multicast is often characterized by sending flow from source to a set of nodes but not all the nodes.

In undirected graphs, the authors in [24] propose an $O(m)$ algorithm for the Bottleneck Steiner Tree BST$^{\downarrow\uparrow}$ problem . The most time consuming subroutine in this algorithm is the computation of a maximal forest. A forest in a graph is a disjoint union of trees. As a consequence, this algorithm runs in $O(m + n \log n)$. In [16], the author propose a very simple alternative to solve BST$^{\downarrow\uparrow}$ in $O(\min\{n^2, m \log\log m\}$ time. This algorithm runs in two successive steps: first, a maximum spanning tree is computed (over the whole graph G). Then, the Steiner nodes (the ones that are not terminals) are scanned in cyclic fashion and removed each time their degree (in the remaining tree) is equal to one:

Algorithm 3.3.4.1 : BST

Input: A weighted undirected graph $G = (V, E, w)$ and a set $T \subseteq V$ of terminals;
Output: A bottleneck Steiner tree $t \in \mathcal{T}^{\downarrow\uparrow}(T)$;

1. Compute a maximum spanning tree $t \in \mathcal{T}^{\uparrow}$ in G;
2. Remove iteratively from t the degree one nodes in $V \setminus T$ (Steiner nodes);

The correctness of the approach is given in the following Lemma 3.3.2.

Lemma 3.3.2. *Consider $t^* \in \mathcal{T}^{\uparrow}$ (a maximum spanning tree) and denote $t^*(T)$ the subtree of t^* spanning the set of terminals $T \subseteq V$. Then $t^*(T) \in \mathcal{T}^{\uparrow\downarrow}(T)$ (in other words, $t^*(T)$ is a -maxmin- bottleneck Steiner tree).*

Proof. First note that $t^*(T)$ is a tree spanning T, i.e., $t^*(T) \in \mathcal{T}(T)$. It follows that:

$$\min_{e \in t^*(T)} w_e \leq \max_{t \in \mathcal{T}(T)} \min_{e \in t} w_e. \tag{3.23}$$

Consider an edge $e^* \in t^*(T)$ where the minimum weight over $t^*(T)$ is reached:

$$w_{e^*} = \min_{e \in t^*(T)} w_e. \tag{3.24}$$

Denote $V = V_1 \oplus V_2$ the partition of V into two subsets obtained when removing the edge e^* from the spanning tree t^*. If we denote $E(V_1, V_2)$ the cut in the graph between V_1 and V_2, we have:

$$w_e^* = \max_{e \in E(V_1, V_2)} w_e, \tag{3.25}$$

because, otherwise, we could replace e^* by the edge where the maximum is reached and obtain a spanning tree of greater weight than t^* (cut property).

Denote $T_1 = V_1 \cap T$ and $T_2 = V_2 \cap T$. By construction (e^* belongs to $t^*(T)$), we have $T_1 \neq \emptyset$ and $T_2 \neq \emptyset$. Every tree spanning T must connect T_1 with T_2 and hence use at least one edge of the cut $E(V_1, V_2)$. It follows that, for each tree $t \in \mathcal{T}(T)$, we have:

$$\min_{e \in t} w_e \leq \max_{e \in E(V_1, V_2)} w_e = w_e^*. \tag{3.26}$$

This is hence also true for a bottleneck tree:

$$\max_{t \in \mathcal{T}(T)} \min_{e \in t} w_e \leq w_e^*. \tag{3.27}$$

By (3.23) and (3.24) , we hence have:

$$\max_{t \in \mathcal{T}(T)} \min_{e \in t} w_e = w_e^*, \tag{3.28}$$

which proves the lemma. □

Combining the second step from Algorithm 3.3.4 with Algorithm 3.3.2.2, we can also solve the BST in directed graphs. The complexity of the corresponding algorithm will remain the same as Algorithm 3.3.2.2.

3.3.5 Full Steiner tree problems

With an additional constraint on each terminal as we defined in the previous section, a Steiner tree is said to be *full Steiner*, if all terminals ($v \in T$) are leaves of the tree (or equivalently, degree one nodes in the tree). If we denote by $\mathcal{T}_\circ(T)$ the set of trees t spanning T and such that $d_T(v) = 1$, for all $v \in T$, then all Steiner tree problems can be transformed into full Steiner tree problems. For instance, the full Steiner tree problem can be described as:

$$\mathrm{FST}^{\downarrow} : \min_{t \in \mathcal{T}_\circ(T)} \operatorname{sum}_{e \in t} w_e. \tag{3.29}$$

In this paper, we are interested in the full bottleneck Steiner tree problem:

$$\mathrm{FBST}^{\downarrow\uparrow} : \min_{t \in \mathcal{T}_\circ(T)} \max_{e \in t} w_e, \tag{3.30}$$

$$\mathrm{FBST}^{\uparrow\downarrow} : \max_{t \in \mathcal{T}_\circ(T)} \min_{e \in t} w_e, \tag{3.31}$$

We denote $\mathcal{T}_\circ^{\downarrow\uparrow}(T)$ and $\mathcal{T}_\circ^{\uparrow\downarrow}(T)$ the set of min-max and max-min full bottleneck Steiner trees, respectively. As stated in the introduction in this chapter, the problem has not been widely studied, and there is only one state-of-the-art algorithm for solving the full bottleneck Steiner tree problem in undirected graphs, but it only works for some special instances. In the following sections, we provide different algorithms to handle this problem in either undirected or directed graphs. Before introducing our algorithms, we will show some preliminary results that may be helpful in describing the oncoming algorithms.

3.4 Preliminary results

In this Section, we recall some simple results (mostly without proofs) concerning the various problems considered. These results will be used in the following Sections.

First of all, in the problem concerned with spanning trees (trees spanning the whole graph), the weight vector can be translated without changing the optimal solutions.

Lemma 3.4.1. *Given any real value Δ, if t^* is an optimal solution of $MT^\downarrow$ or $MT^\uparrow$ in the graph G with weights $(w_e)_{e\in E}$, then it is also a solution of the same problem in the graph G with weights $(w_e+\Delta)_{e\in E}$. The optimal value is grown by the quantity $\Delta(|V|-1)$.*

An even stronger result holds for bottleneck problems, since it is only the order on the edges induced by the weights that matters.

Lemma 3.4.2. *Consider any mapping $\sigma : E \to E$ such that, for any $(e,e') \in E^2$, if $w_e < w_{e'}$, then $\sigma(w_e) < \sigma(w_{e'})$. If t^* is an optimal solution of a bottleneck problem in the graph G with weights $(w_e)_{e\in E}$, then it is also a solution of the same problem in the graph G with weights $(\sigma(w_e))_{e\in E}$. The optimal value w^* becomes $\sigma(w^*)$.*

Similar results are obtained when weights or the order induced by weights is reversed. In this case, the optimal solutions are changed into solutions of the "reversed" problem (where max and min are exchanged). For instance:

Lemma 3.4.3. *If t^* is an optimal solution of $MT^\downarrow$ in the graph G with weights $(w_e)_{e\in E}$, then it is also a solution of the problem $MT^\uparrow$ in the graph G with weights $(-w_e)_{e\in E}$.*

Another interesting (although straightforward) result states that a bottleneck problem is unchanged if edges with sufficiently small or large weight are added to the graph.

Lemma 3.4.4. *Consider a weighted graph $G = (V, E, w)$ where w_{min} and w_{max} denote the smallest and largest weights.*

1. *If t^* is an optimal solution of $BT^{\downarrow\uparrow}$ in the graph G, then it is also a solution in the extended (complete) graph $H = K_{|V|}$ where each additional edge has the weight $w_{max} + 1$.*

2. *If t^* is an optimal solution of $BT^{\uparrow\downarrow}$ in the graph G, then it is also a solution in the extended (complete) graph $H = K_{|V|}$ where each additional edge has the weight $w_{min} - 1$.*

3.5 An $O(|S|^2|T|)$ Algorithm for the Bottleneck Full Steiner Tree Problem in an Undirected Graph

In a multicast network, it is very often the case that the client of the service (in the Steiner tree context, the terminals) are end-nodes of the network (access nodes). As such, they only act as receivers and do not forward the traffic further to other nodes. Using Steiner trees to model a multicast network, it would hence be necessary to add the constraint that the terminals should be leaf-nodes of the tree. The full bottleneck Steiner tree problem seems hence a good candidate to model multicast networks. A $O(|E| \log |E|)$ algorithm was proposed in [15] for the min-max version $\text{FBST}^{\uparrow\downarrow}$. The proof of correctness of this algorithm requires some assumptions on the graph: the weights must be all strictly different and the graph should be complete.

In this section, we present an algorithm which can be applied to general network topologies, in other words, our algorithm does not require the two specific assumptions anymore. In order to fit the algorithm with our need in the context of multicast networks, we adapt the algorithm and all related results to the max-min case: indeed, our concern in multicast network, is to maximize the end-to-end throughput, and hence to maximize the bottleneck edge (the edge with minimum capacity).

3.5.1 Feasibility of the instances

First, we should note that any instance of weighted graph is not necessarily feasible for any kind of Full Steiner Tree (FST) problem. The instance depicted in Figure 3.1 is a simple example of graph that does not contain any FST.

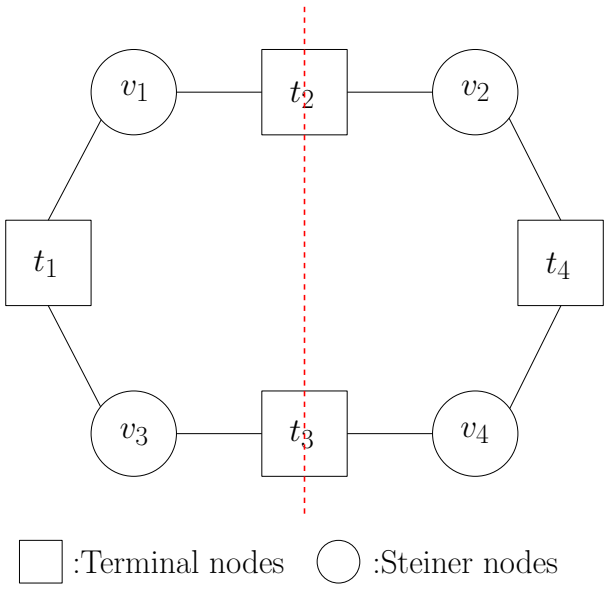


Figure 3.1: Infeasible instance for a Full Steiner Tree Problem

The infeasibility can be assessed by a simple observation: consider the node cut $\{t^2, t^3\}$. The graph obtained when removing these two terminals contains two connected components, each one containing one other terminal. Since the Steiner tree must connect all terminals, at least one of the nodes of the cut must be used to connect both components, and as a consequence, at least one terminal must be of degree more than one. This property can be generalized:

Theorem 3.5.1. *A graph G does not admit full Steiner tree if there exists a vertex cut set $C \subset T$, such that, at least two connected components of the resulting graph, each one containing at least one terminal.*

Proof. If there exists such a vertex cut, call V_1 and V_2 the two connected components in $G \setminus C$ and assume V_i contains t_k, for $k = 1, 2$. Then, to obtain a Steiner tree in the original graph, there must exist a path between t_1 and t_2. By construction, this path must necessarily go through one node $t \in C$. As a result, at least one terminal $t \in C$ cannot be of degree one. □

This condition is difficult to be applied in practice because it is based on a combinatorial number of cuts. When the size of the graph grows, the set of node

cuts increase exponentially, but there indeed exists the polynomial time bounded algorithm for checking the feasibility on a given graph. We will present a reasonable one after making some necessary analysis on the graph.

3.5.2 An Polynomial Time algorithm for Examining Feasibility

Given an undirected graph G, we first tailor it to a so-called *tight graph*. A tight graph is a graph that does not contain any degree-one Steiner nodes. A degree-one Steiner node indicates there is only one link attached to the corresponding Steiner node. The tailoring process can be done by pruning iteratively the degree-one Steiner nodes $v \in S$ from the original graph G. For the full Steiner tree problem, the tailor operation does not influence the final solution, but it can help avoid unnecessary operations during the analysis and the implementation of the corresponding algorithms.

Lemma 3.5.1. *The degree-one Steiner nodes in the original graph are redundant nodes to build a full Steiner tree.*

Proof. Assume that there exists a full Steiner tree $\mathcal{T}_\circ(T)$ where $d_T(t_k) = 1$, the Steiner nodes $v \in S$ in this tree always belong to at least one *terminal path*. A terminal path is a path that has two terminal nodes as the end nodes of the path. It indicates that in a given graph G, the Steiner nodes which lie on such paths are effective candidates for building a full Steiner tree. On the contrary, if a Steiner node lies only on a path that has only one terminal node as the end node or has no terminal nodes as the end nodes, they will eventually be delete when the full Steiner tree is found. The degree-one Steiner nodes are indeed these kinds of nodes and can be easily found in graph G. □

After pruning the degree-one Steiner nodes from G, we subtract all the terminal-Steiner edges from the tight graph G and examine the feature of the subgraphs of G. A terminal-Steiner edge is an edge that directly links one terminal node and one Steiner node. We still use G to represent the tight graph, since the original graph and the tight graph produce the same solution. The residual graphs without the terminal-Steiner edges are classified into two classes, since the performances and the complexities of the algorithms that are proposed in the following for these two sorts of subgraphs are slightly different.

- **First Class:** the residual graph is a single connected component. It can be a path, a tree, a subgraph, or even a single point.

- **Second Class:** the residual graph is a set of disconnected components. The single component in the set has the same definition as the first class.

If the residual graph belongs to the first class, we can easily build a feasible full bottleneck Steiner tree by first implementing a maximum spanning tree algorithm on the residual graph and then arbitrarily add terminal-Steiner edges to the spanning tree until all the terminals are attached to it.

However, the feasibility is not that obvious when we have a case belonging to the second class residual graph, but we will provide a polynomial time tractable algorithm for resolving it. It should be noticed that having a feasible solution in this case suggests that all the terminals could be connected to at least one component of the set of disconnected components. Otherwise, the problem apparently has a terminal cut which follows the argument in Theorem 3.5.1. According to this context, checking the feasibility on each component in the residual graph is an appropriate alternative for judging the feasibility of the original problem rather than finding the terminal cuts.

The algorithm for checking whether a given graph has full Steiner trees for certain terminal nodes is displayed in Algorithm **FBST-check**. It must be noted that the **continue** statement in the pseudo-code has the same meaning as in $C++$. EXIT_SUCCESS and EXIT_FAILURE are the instructions to exit the program with feasible and infeasible indications, respectively. This resulting algorithm runs in polynomial time, and the corresponding complexity proof is given in Lemma 3.5.2.

Lemma 3.5.2. *The complexity of* **FBST-check** *is bounded by* $O(|S||T|\log|T|)$.

Proof. The first **for** loop and the while loop allow the algorithm to traverse all the nodes $v \in S$ in the residual graph (without terminal-Steiner edges), therefore it runs in $O(|S|)$ without consideration of any operations inside the loops. The nested **for** loop between instructions 4 and 14 visits all the terminals attached to a specific Steiner node. Regarding the worse scenarios, every Steiner node may connect to all the terminal nodes, in other words, the loop may traverse all the terminals, and therefore the running time of the loop is bounded by $O(|T|)$.

Moreover, it ought to be noticed that Instruction 5 actually runs a hidden subalgorithm **find** which searches a given element in a set. The best search algorithm, such as binary search algorithm, runs in $O(\log n)$, where n denotes the number

of elements in a set. In our case, the largest possible set of T_c is equal to T, as a result, the search algorithm is then is bounded by $O(\log |T|)$. To sum up, the combination of all the loops and the sub-algorithm **find** let the entire algorithm **FBST-check** be bounded by $O(|S||T| \log |T|)$. □

Algorithm 3.5.2.1 : FBST-check

Input: A residual graph $G \setminus \{T, E(T)\}$ which has m components, $G_i, i = 1, 2, \ldots, m$. An empty set T_c and a counter $j = 0$. The number of terminals $|T|$.

Output: Problem's feasibility.

```
1. for all G_i, i = 1, 2, ..., m do
2.   while G_i ≠ ∅ do
3.     Select v ∈ G_i;
4.     for all t ∈ T(v) do
5.       Check t whether in the T_c;
6.       if t ∈ T_c then
7.         continue;
8.       else
9.         Add t in T_C, and increase j by 1;
10.      end if
11.      if j = |T| then
12.        EXIT_SUCCESS;
13.      end if
14.    end for
15.    Remove v from G_i;
16.  end while
17.  if i = m then
18.    EXIT_FAILURE;
19.  end if
20.  Reset T_c and counter j;
21. end for
```

Once we confirm that a given graph G is capable to build a full bottleneck Steiner tree for given terminals, the next question is how to find an optimal tree in $\mathcal{T}_\circ^{\downarrow\uparrow}(T)$.

3.5.3 The Core Sub-Algorithm in Finding $\mathcal{T}_\circ^{\downarrow\uparrow}(T)$

Before giving the algorithm that finds a $\mathcal{T}_\circ^{\downarrow\downarrow}(T)$, we begin by analyzing a simple algorithm which finds a proper feasible solution. The analysis is based on the graphs having a first class residual graph, in other words, the analysis serves a single connected subgraph. We denote the connected component by $G' = (S, E(S))$. For those graphs that have a second class residual graph, the analysis can be easily adapted, since the second class residual graph consists of multiple single

connected component. The simple algorithm helps to develop the key idea for the design of our ultimate algorithm. It is an algorithm which takes the maximum spanning tree algorithm as a core sub-algorithm, so we simply call it MT-based algorithm. It consists of two steps: in the first step, we run a maximum spanning tree algorithm on G'; in the second step, we assign the maximum $E(t), \forall t \in T$ to the tree that spans the Steiner nodes. A simple example in Fig. 3.2 explains why this algorithm does not always achieve an optimal solution. The figure shows the last operation that attaches the last terminal t_1 to the tree consisting of several solid lines. We assume that the current bottleneck before terminal t_1 joins in is the edge $e_{\{B,C\}}$ with a weight $w_{\{B,C\}} = 11$. According to the second step of the MT-based algorithm, the maximum weight in $E(t_1)$ is $e_{\{t_1,C\}}$, and it will be chosen. As a result, the bottleneck of the full Steiner tree will be $w_{opt} = 11$. However, if there is a chance to choose the edge $e_{\{t_1,B\}}$ alternatively, the optimal value increases to $w'_{opt} = 12$, which is better than the previous result. This algorithm does not guarantee a good solution though it is easy to implement. For example, suppose that the weight $w_{\{B,C\}} = 11$ is replaced by a very small value $w_{\{B,C\}} = 1$, then the bottleneck will suddenly drop to $w_{opt} = 1$.

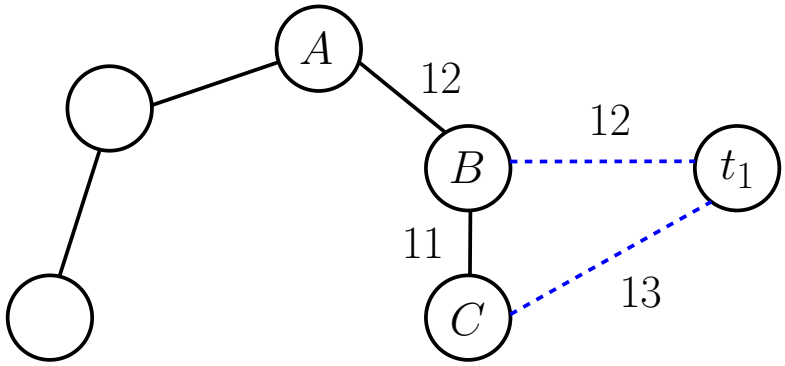


Figure 3.2: An example of non-optimal solution

The Steiner nodes' spanning tree is a proper frame to begin finding the optimal tree. All the terminals are always attached to a subtree which only consists of Steiner nodes in a full Steiner tree. This tree is denoted by $\mathcal{T}_s$ where $\mathcal{T}_s \in G'$. Randomly choosing a tree in G' could hardly guarantee the connectivities to all the terminals. In addition to that, if the bottleneck is an edge $e \in E(S)$, a random tree also hardly promises to include the edge e. But there is always a subtree $\mathcal{T}_s$ on maximum spanning tree $\mathcal{T}_s^{\uparrow}$ that is a proper candidate/stem to form an optimal full Steiner tree. This argument can be easily approved. First, as a spanning tree, $\mathcal{T}_s^{\uparrow}$ spans all the Steiner nodes, which implies that the tree $\mathcal{T}_s^{\uparrow}$

is capable to connect to all the terminals when the original problem is feasible. Second, since a subtree on a maximum spanning tree is also a bottleneck Steiner tree in graph G' for spanning those nodes belonging to the subtree, it guarantees that in G', the bottleneck of the other subtrees formed by a certain set of Steiner nodes will never be better than the one provided in $\mathcal{T}_s^{\uparrow}$.

Instead of searching a subtree $\mathcal{T}_s$ from $\mathcal{T}_s^{\uparrow}$, we try to attach proper terminal-Steiner edges to $\mathcal{T}_s^{\uparrow}$ in order to accomplish the final tree construction. The task can be resolved properly by using the next algorithm we provide. We name the algorithm **FBST-CORE**, since it tackles the core issue for building the full bottleneck Steiner tree.

Assume that there is a Steiner-terminal edge $e \in E(S,T)$ that directly connects to terminal t. The key idea of the algorithm **FBST-CORE** is to update the bottleneck information between t and all the Steiner nodes on the tree T_s through this particular edge e. Assume that there are multiple Steiner-terminal edges linking to t, the information needs to be updated multiple times. The same procedure for the other terminals is performed independently, which means that the bottleneck information for different terminals is stored separately. After the update process, we compare the bottleneck information achieved by different terminals labeled on each Steiner node. We first pick up the minimum value from the information that corresponds to different terminals on each Steiner node, then from the set of Steiner nodes S_T that have direct connections to the terminals, we choose the maximum value from the minimum values we just found. The maximum value will be the optimal bottleneck that can be achieved for the full bottleneck Steiner tree problem. It should be noted that the procedure mentioned above is under the assumption that the residual graph of a given original graph belongs to the first class. For the graphs that are classified into the second class, the algorithm **FBST-CORE** has to be run in all the potential connected components that have feasible solutions. By comparing the optimal bottleneck found independently among the connected components, we choose the connected component that has the maximum bottleneck to construct a $\mathcal{T}_{\circ}^{\downarrow\uparrow}(T)$.

The details of the algorithm will be given after the introduction of some essential notation and parameters with their initializations. We denote the set of Steiner nodes which have direct connections to at least one terminal by S_T, and the rest of Steiner nodes by $S_{\overline{T}}$. We create $|T|$ triples (t,w,v) for $S_{\overline{T}}$; $|T(v)|$ quadruples (t,w,v,u) and $|T|-|T(v)|$ triples for $v \in S_T$. As a result, there are $|T|$ tuples on

each Steiner node, therefore there are $|S|\cdot|T|$ tuples in total. t, w, v, u successively denote the terminal, the weight of bottleneck, the predecessor of the current one in a bottleneck tree, and the statement of whether the tuple is used. For $v \in S_{\overline{T}}$, each triple is reserved for a specific terminal, therefore the terminal is assigned to the first element of each triple accordingly. For $v \in S_T$, the terminals $T(v)$ are assigned to t in the quadruples, and the other terminals are assigned to t in the triples. For an edge $(t, v) \in E(t)$, its weight is assigned to w in the quadruples with the same t on the node v. All the other w and v in the tuples that we have not yet mentioned are uniformly set to **null**, which is equivalent to 0, 'none' or 'empty'. Moreover, u in each quadruple is always set to **unused** in the beginning of the algorithm.

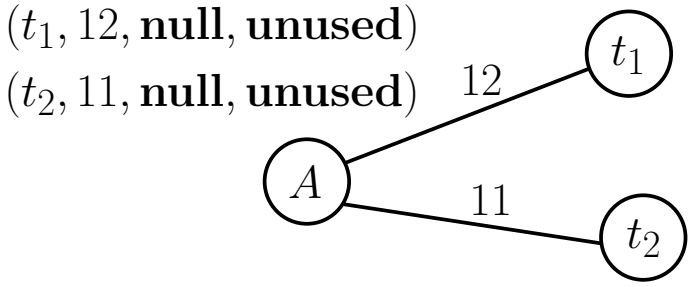


Figure 3.3: An example of initializing some quadruples

At the beginning of the algorithm, all $v \in S_T$ are marked, and the algorithm arbitrarily starts from one of them. Each **unused** quadruple will turn to **used**, when either it starts to update the other tuples in the tree $\mathcal{T}_s^{\uparrow}$ that has the same t on the other Steiner nodes, or it is updated by other tuples. For the worst scenario, the updating process has to go through all the nodes on the spanning tree $\mathcal{T}_s^{\uparrow}$. Each $v \in S_T$ is set to unmarked until all the quadruples on that node are **used**. The algorithm **FBST-CORE** finishes when all $v \in S_T$ are unmarked.

The information update in the tuples follows the rules below: Assume two nodes $A, B \in S$, and we try to update the tuple for terminal t on node B according to the current information on node A and edge (A, B). Then

$$w_B^t = \max\{w_B^t, \min\{w_A^t, w_{(A,B)}\}\} \tag{3.32}$$

if $w_B^t = \min\{w_A^t, w_{(A,B)}\}$, then $v_B = A$. Moreover, if the tuple is a quadruple, and we assign u to **used** if u is **unused**. If $w_B^t \neq \min\{w_A^t, w_{(A,B)}\}$, v_B does not update.

The pseudo-code of the algorithm **FBST-CORE** is presented in Algorithm 3.5.3.1. In order to show that each update for the next tuple is according to the same terminal, we set an extra superscript for each element w, v, and u in every tuple, for instance, w^t, v^t, and u^t.

Algorithm 3.5.3.1 : FBST-CORE

Input: A Steiner nodes' maximum spanning tree residual graph $\mathcal{T}_s^{\uparrow}$, $|S| \cdot |T|$ tuples with appropriate initialization. An empty candidate set X;
Output: Updated $|S| \cdot |T|$ tuples;

1. **for** all $p \in S_T$ **do**
2. Select $p \in S_T$ and put it in a candidate set X;
3. **while** $X \neq \emptyset$ **do**
4. Select $p \in X$;
5. **for** all $t \in T(p)$ **do**
6. Set $u_p^t =$ **used**;
7. **for** $q \in S(v)$ **do**
8. $w_q^t = \max\{w_q^t, \min\{w_p^t, w_{(p,q)}\}\}$;
9. **if** $w_q^t = \min\{w_p^t, w_{(p,q)}\}$ **then**
10. $v_q^t = p$ and put q in the X;
11. **if** there is u_q^t and $u_q^t ==$ **unused then**
12. Set $u_q^t =$ **used**;
13. **end if**
14. **end if**
15. **end for**
16. **end for**
17. **end while**
18. **end for**

By obtaining the information from the updated tuples, we know the bottleneck value $w_{opt} = \max_{v \in S_T}\{\min_{t \in T}\{w_v^t\}\}$. Assume there is a node $v \in S_T$ where $w_v = w_{opt}$. From the definition of w_{opt} we know that, for each terminal, either there is a bottleneck path between a terminal and node v in which the value of the bottleneck is greater than or equal to w_{opt}, or a terminal t has a direct connection to v by using $E(t)$, the weight of which is no less than w_{opt}.

After finding the optimal bottleneck value, there are four simple steps to finalize the tree construction: first, we set the value $w_v = \min_{t \in T}\{w_v^t\}$ for each $v \in \mathcal{T}_s^{\uparrow}$; second, we collect the nodes with the weights $w_v \geq w_{opt}$ in a node set S_s; third, we prune the tree $\mathcal{T}_s^{\uparrow}$ by subtracting the degree-one Steiner node $v \in S \setminus S_s$ and obtain the tree T_s; fourth, we randomly choose edges $e \in E(t)$ where $e \geq w_{opt}$ for every t from the nodes $v \in S_s \cap S_T, w_v = w_{opt}$.

In order to judge the complexity of the whole procedure that finds the full bottleneck Steiner tree, we start by analyzing the algorithm **FBST-CORE**.

Lemma 3.5.3. *The complexity of* **FBST-CORE** *is bounded by* $O(|S|^2|T|)$*, where* $|S|$*, and* $|T|$ *denote the number of Steiner nodes and terminals, respectively.*

Proof. In the worst case, for certain terminal t in the *for* loop between 5 and 16, $|S|$ Steiner nodes have to update their information, in other words, the algorithm will traverse the whole spanning tree $\mathcal{T}_s^{\uparrow}$. Since each instruction in this loop is fundamental operation which is bounded by $O(1)$, then this loop can be bounded by $O(|S-1|)$. Assume that each terminal may connect to $|S(t)|$ nodes, so the algorithm **FBST-CORE** will traverse the tree $\mathcal{T}_s^{\uparrow}$ n times, where $n = \sum_{t\in T} |S(t)|$, which is equal to the amount of quadruples. Considering the worst case, such as full mesh networks, where $n = |T| \cdot |S|$, the algorithm is bounded by $O(|S||S-1||T|) = O(|S|^2|T|)$. □

Algorithm 3.5.3.2 : FBST

Input: An undirected graph $G = (V, E)$;
Output: Optimal tree $\mathcal{T}_{\circ}^{\downarrow\downarrow}(T)$ or Infeasibility;

1. Check the connectivity of $G \setminus \{T, E(T)\}$;
2. **if** $G \setminus \{T, E(T)\}$ is in the first class **then**
3. Implement maximum spanning tree algorithm on $G \setminus \{T, E(T)\}$ and obtain $\mathcal{T}_s^{\uparrow}$;
4. Implement algorithm **FBST-CORE**;
5. Construct the final tree from updated information of the tuples from algorithm **FBST-CORE**;
6. **else**
7. Find subgraphs in $G \setminus \{T, E(T)\}$;
8. Implement algorithm **FBST-check**;
9. **if** The problem is feasible **then**
10. Implement algorithm **FBST-CORE** in all feasible components;
11. Find the best component and construct the final tree from it;
12. **else**
13. The problem is infeasible;
14. **end if**
15. **end if**

The entire procedure of finding full bottleneck Steiner tree could be summarized in the Algorithm 3.5.3.2. In fact, after judging the connectivity for the residual graph $G \setminus \{T, E(T)\}$, the algorithm is split into two subroutines for different classifications of the residual graphs. For the graphs having a first class residual graph, the subroutine is dominated by instruction 4 in Algorithm 3.5.3.2, and the complexity of this branch is bounded by $O(|S|^2|T|)$. Similarly for the graphs

belonging to the second class, since the algorithm **FBST-CORE** must be run k times, where k is equivalent to the number of feasible connected components in the residual graph. For the worst scenarios, k may be equal to $|S|$, therefore this subroutine is bounded by $O(|S|^3|T|)$. As a result, the complexity of the entire procedure runs at $O(|S|^3|T|)$.

3.6 An $O(|E| \log |E|)$ Algorithm for the full Bottleneck Steiner Tree Problem in Undirected Graph

This algorithm is very similar to Kruskal's algorithm for computing minimum (or maximum) spanning trees. It requires two data structures:

- An integer label ℓ_v associated with each Steiner node $v \in S$;
- A list (or a set) $\mathcal{L}_v$ associated with each terminal node $v \in T$.

We first make the same assumption as in [15] that all edge weights are strictly different. We will then see how this assumption can also be relaxed by modifying our algorithm.

As in Kruskal's algorithm, the labels are used to identify nodes within a same connected component and hence avoid forming cycles. When adding an edge, labels are updated and in order to minimize the number of updates, one can also maintain, for each Steiner node $v \in S$, an integer value sz_v giving the current size of the connected component containing node v. Finally, we assume that the edges have been ranked in decreasing order and are stored in that order in a list $\mathcal{L}_E^{\downarrow}$.

Our algorithm to compute a max-min Bottleneck Full Steiner Tree (BFST) is formalized in Algorithm 3.6.0.3.

As mentioned before, the labels ℓ_v are used to identify Steiner nodes within a same connected component. The label lists $\mathcal{L}_v$ indicate to which connected component (defined only among the Steiner nodes) each terminal is currently connected. The algorithm stops as soon as there is a connected component to which all terminals are connected. The stopping criteria $\mathcal{L}_E^{\downarrow} = \emptyset$ is never reached in practice, except

Algorithm 3.6.0.3 : FBST2

Input: A weighted undirected graph $G = (V, E, w)$ and a set T of terminals;
Output: A max-min tree τ spanning T such that all terminals in T have degree one;

1. Initialize $\ell_v = v$ for all $v \in S$, $\mathcal{L}_v = \emptyset$ for all $v \in T$ and $\tau = \emptyset$;
2. **repeat**
3. Remove the first edge from $\mathcal{L}_E^{\downarrow} \rightarrow e = (a, b)$;
4. **if** $e \in E(S)$ and $\ell_a \neq \ell_b$ **then**
5. add e to τ;
6. set to ℓ_b the label of all nodes v such that $\ell_v = \ell_a$ (without loss of generality, assuming $d_a < d_b$);
7. replace ℓ_a with ℓ_b in all lists $\mathcal{L}_v$ containing ℓ_a;
8. **else if** $e = (a, b) \in E(S, T)$ **then**
9. add e to τ;
10. insert label ℓ_a in list $\mathcal{L}_b$ (assuming w.l.o.g. $a \in S, b \in T$);
11. **end if**
12. **until** $\cap_{v \in T} \mathcal{L}_v \neq \emptyset$ **or** $\mathcal{L}_E^{\downarrow} = \emptyset$;
13. run algorithm **TRIM-TREE** to obtain the final tree τ;

if the initial graph is already a tree.

After the stopping criterion (at step 12), the algorithm has already identified (i) a bottleneck edge and (ii) the bottleneck weight (weight of any bottleneck edge). The purpose of the additional step invoking algorithm TRIM-TREE is to trim unnecessary edges in τ in order to obtain a minimal (in the sense of inclusion) Steiner tree.

Algorithm 3.6.0.4 : TRIM-TREE

Input: A connected component τ_{in} of the graph $G = (V, E)$ containing all terminals $t \in T$;
Output: A tree τ_{out} spanning T such that all terminals in T have degree one;

1. $\tau \leftarrow \tau_{in}$;
2. **for** all terminals $t \in T$ **do**
3. find in $E(t) \cap \tau$ (stored in $\mathcal{L}_t$) the edge e_t^{max} of maximum weight;
4. remove from τ all edges in $E(t) \cap \tau$ except e_t^{max};
5. **end for**
6. **repeat**
7. remove from τ all edges $\{u, v\} \in E(S)$ such that u or v is of degree one;
8. **until** there are no more Steiner nodes of degree one;

In the algorithm TRIM-TREE, only the edge of maximum weight connecting each terminal to the identified connected component is kept. As used in [16] to obtain a Bottleneck Steiner tree, all degree one Steiner nodes are progressively

removed from the tree (more precisely, its adjacent edge is removed from the edge list).

The proof of correctness is simple: after the stopping criterion is reached (at step 12), there is only one connected component in the subgraph $G_S = (S, E(S))$ that is also connected to every terminal in T. All edges in G_S form a forest composed of edges of maximum weight (the edges missing are the ones creating cycles). To connect all terminals to another connected component, one would have to introduce at least one other edge of weight strictly smaller than the current bottleneck. Among all edges connecting the terminals to the same connected component, it suffices to keep one (so we choose the one with maximum weight).

The time complexity of the algorithm is basically the same as the one of Kruskal's algorithm. The main differences are (i) the algorithm might stop before the subgraph in G_S is a tree (it indeed suffices to have one connected component already connected to all terminals) and (ii) in the stopping criterion, one has to find the intersection of n_T sets. If the sets are stored as sorted lists, this can be achieved in $O(|T|)$ time. The additional steps in TRIM-TREE run respectively in constant and $O(|S|)$ time. As for the standard Kruskal's algorithm, the initial phase where edges are sorted in decreasing order of their weight requires $O(|E| \log |E|)$ time which is hence still the dominating time complexity.

Lemma 3.6.1. *Algorithm FBST2 runs in $O(|E| \log |E|)$ time.*

The algorithm **FBST2** is easy to implement and very efficient. Figure 3.4 shows a result obtained on 200 node graph. As expected, some Steiner nodes are not used and the result is very far from a star.

3.6.1 Edges with the same weight

In algorithm FBST2, we have assumed that all edge weights are strictly different. If, at a given step, there are several edges of $E(S)$ having the same weight, then the decomposition of the subgraph of Steiner nodes into connected components is not unique. And according to the existing edges in $E(S, T)$ one decomposition might allow to stop the algorithm and not another.

For instance, on the graph of Figure 3.5, we see that there are two edges carrying the weight 8. If the edge (v_2, v_4) is added, the algorithm stops (all terminals are connected to the same connected component). If the edge (v_3, v_5) is added, the

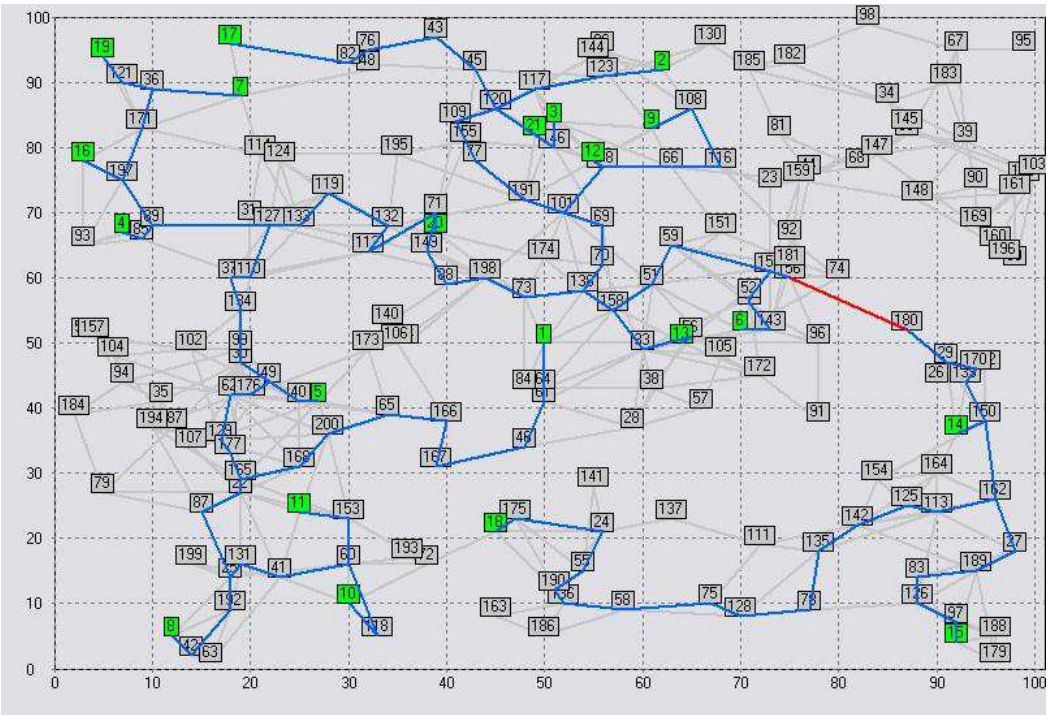


Figure 3.4: Euclidean min-max full bottleneck Steiner tree instance: a random graph with 200 nodes, 500 edges and 20 terminals

algorithm does not stop immediately. However, at the next step, the edge (v_2, v_4) is going to be added and the algorithm will stop with the same bottleneck value. As a result, it is easy to see that Algorithm FBST2 remains valid, even if there are several edges with the same weight.

3.7 More efficient algorithms

As observed in [12, 32], bottleneck or max-min problems are often easier to solve than their max-sum counterparts. The algorithms proposed in these papers focus on identifying the bottleneck value w^* by exploring wisely the set of potential values $\{w_e\}_{e \in E}$. Note that, if one was able to "guess" the optimal value w^*, the optimality of this value could be checked in $O(m)$ time. Indeed, it suffices to build the subgraphs $G^{\leq}_{w^*}$ and $G^{<}_{w^*}$ defined by:

$$G^{\leq}_{w^*} = (V, \{e \in E : w_e \leq w^*\}), \tag{3.33}$$

$$G^{<}_{w^*} = (V, \{e \in E : w_e < w^*\}). \tag{3.34}$$

A simple depth-first exploration checking that $G^{\leq}_{w^*}$ is connected and $G^{<}_{w^*}$ is not connected suffices to prove that w^* is the bottleneck value. The same reasoning

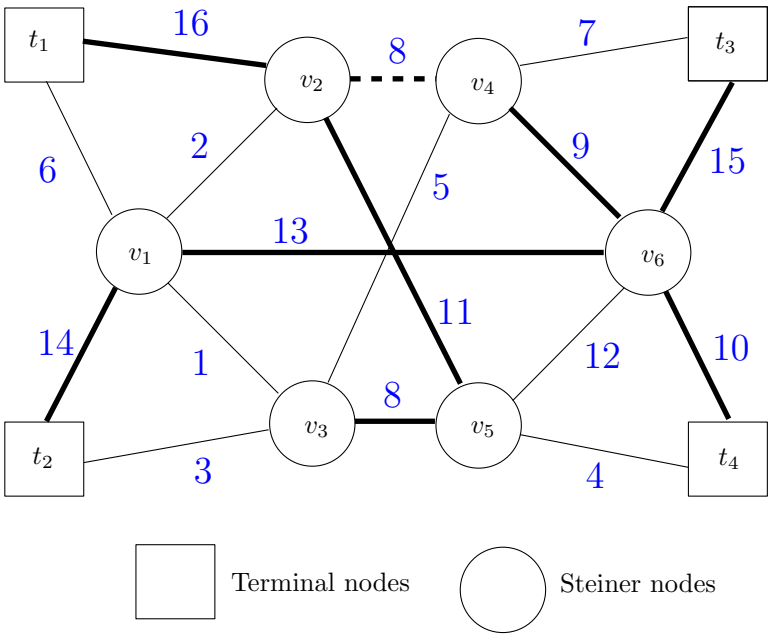


Figure 3.5: Instance with two edges with same weight

can be used to identify a bottleneck edge in a full Steiner tree. Indeed, it suffices to check that there is a component of $G^{\leq}_{w^*}$ connecting all terminals and that there is no such component in $G^{<}_{w^*}$ to assess that w^* is a bottleneck edge (see, for instance, algorithm EXPLORE).

Algorithm 3.7.0.1 : EXPLORE

Input: A graph or subgraph $G = (V, E)$ and a subset of terminals $T \subset V$;
Output: TRUE is G contains a component connecting all terminals $t \in T$;

1. choose a starting terminal $t_0 \in T$ of minimal degree;
2. let $T_0 = \gamma(t_0)$ and forall $v \in T_0$ remove the edge $\{t_0, v\}$ from E;
3. **for** all $v_0 \in T_0$ **do**
4. let $V_M = \{t_0, v_0\}$;
5. **repeat**
6. **for** all $v \in \gamma(V_M)$ **do**
7. add v to V_M;
8. **end for**
9. **until** (i) $T \subseteq V_M$ **or** (ii) $\delta(V_M) = \emptyset$;
10. if (i) then return TRUE;
11. **end for**

The approach of [12] was extended in [32] for the case of a directed graph where one source terminal s should be connected to all other terminals $t \in T \setminus \{s\}$ with a directed path. Again, the algorithm in [32] can easily be adapted to handle the case of a bottleneck full Steiner tree: in the connectivity criterion, where an

exploration of the sub-graph is started from the source node s (note that here, we do not restrict the degree of the source node), once a terminal is reached, the exploration is stopped at this node and the same trimming algorithm can be applied in order to remove unnecessary edges. As a result, the bottleneck full Steiner tree problem can be solved in a directed graph in $O(|E| \log^* |V|)$ time (where $\log^*$ is the iterated logarithm).

3.8 Conclusion

In this chapter, our concern is the maximum throughput that can be achieved, within a given capacitated network, using a multicast tree to send data to end-users. We review some network problems that could help to model multicast networks in telecommunication field. We point out how to find a best single multicast tree, in terms of end-to-end throughput by using bottleneck version of network flow problems. We focus on the bottleneck Steiner tree problem and the full bottleneck Steiner tree problem. These two special Steiner tree problems can model different multicast networks when single tree is used, depending whether the clients of the multicast service can or cannot forward the data to other clients. The bottleneck Steiner tree problem can be efficiently solved by modified version of Kruskal's algorithm or bottleneck shortest path problem in undirected graphs or directed graphs. We have proposed two algorithms to tackle the full bottleneck Steiner tree problem in general random graphs. The most efficient algorithm we find is a Kruskal-based $O(|E| \log |E|)$ algorithm. We also show how to adapt even more efficient algorithms from the literature to handle the bottleneck and bottleneck full Steiner tree problems. The work in this chapter can be considered as a foundation of the research in the next chapter, which focuses on comparing the end-to-end throughput between multicast network and network coded networks.

Chapter 4

Investigation on Maximum Throughput

As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.

-Albert Einstein (1879-1955)

4.1 Summary

Network coding has been shown to be the solution that allows to reach the theoretical maximum throughput in a capacitated telecommunication network [5]. It has also been shown to be a very appealing and practical alternative to routing-based approaches to send traffic from sources (servers) to terminals (clients) for many different applications. However, the initial theoretical claim of throughput benefit remains relatively unclear, mainly because the multicast throughput maximization problem is difficult to solve (it is closely related to the fractional Steiner tree packing problem which is NP-hard). In this chapter, we show that these optimization problems are still tractable even for instances with a significant size (up to 50 nodes and 300 edges). We also propose and solve the multicast maximum throughput problem with an additional constraint on the number of multicast trees. We apply our algorithms on large sets of randomly generated instances, mainly based on bidirected graphs, because they are the most relevant to model fixed telecommunication infrastructures. The main re-

sult of our intensive experimental study is that, in practice, network coding does not increase throughput compared to traditional multicast. Instances showing a throughput gain can only be generated somewhat artificially by imposing some structure or trying to maximize the throughput gap. However, when we limit the number of multicast trees, then, most of the times, very significant throughput gaps appeared. Since management constraints often impose on network administrators a very limited use of multicast trees, network coding appears clearly as a very attractive alternative for delivering content to customers.

4.2 Introduction

Network coding has been first introduced as a nice solution allowing to reach a theoretical upper-bound on the throughput in multicast networks [5]. Given a capacitated network with a source node and a set of terminal nodes, this theoretical upper-bound refers to the global throughput that would be achieved if the stream between the source and each terminal node could use all the available resources (capacities), regardless of the other streams. The maximum throughput between a source and a destination can hence be obtained as the optimal solution of the well-known Maximum-Flow problem [28].

Since then, network coding [41] has been considerably studied, both from the theoretical and the practical sides [59]. As a natural extension to the field of coding theory, several efficient ways to generate coding schemes and decoding algorithms have been proposed [42, 43, 44, 65]. Network coding applications have been proposed, and sometimes tested in various networks, in various places within the networks, ranging from the application layer to the physical layer, and for various applications [56, 72, 84].

Because of the original statement that network coding permits to reach the maximum possible throughput, a special emphasis has been laid on theoretical and empirical throughput evaluations, essentially focused on a comparison with standard multicast routing [79]. Recall that multicast protocols are designed for services where several users require the same content at the same time (live TV for instance). Multicast (one-to-many) is hence opposed to traditional unicast (one-to-one). The key feature of multicast protocols is to allow some well-chosen network nodes to replicate data towards several outgoing links in order to alleviate network load. Alternatively, network coding (associated with multicast or

not) allows network nodes to combine several data received potentially from different incoming links towards one or several outgoing links. The famous *Butterfly* example is often cited to illustrate the benefit of network coding. We propose another example where the network coding throughput of 3 (Figure 4.1) is larger than the multicast throughput of 2.666 (Figure 4.2).

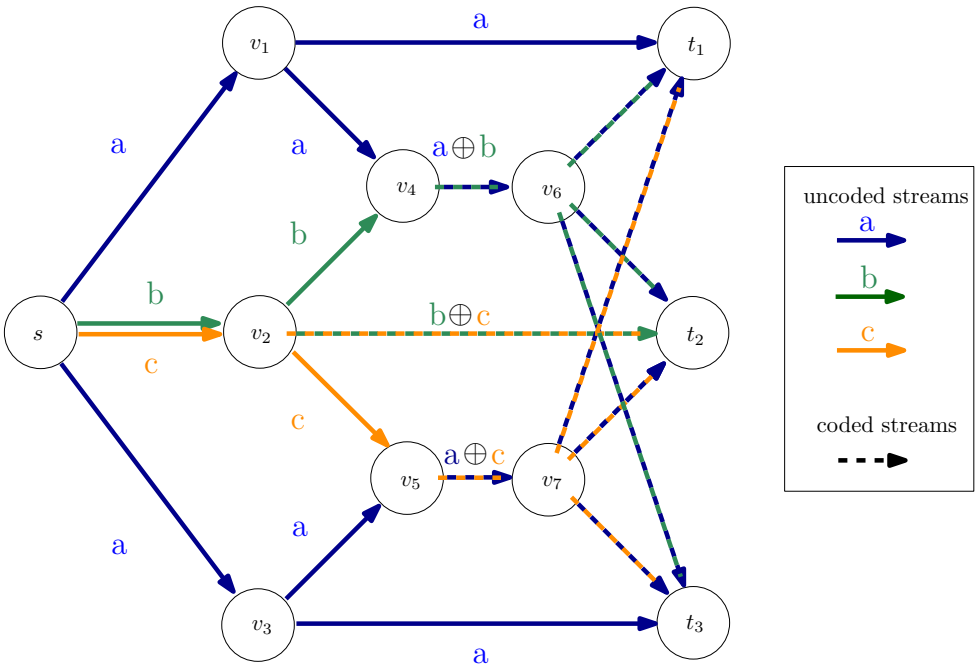


Figure 4.1: Optimal solution with Network Coding (NC) in a butterfly network: throughput = 3, 3 streams a, b and c can be transmitted to each terminal.

Many papers have been devoted to investigate this difference in throughput over different instances types and sets. A main result states that the relative gap in throughput is never larger than 2 is proposed in [67]. When tackling the issue of comparing network coding with multicast throughput, one has to face optimization problems from graph theory. The first one, the maximum-flow problem, is very well-known and can be solved very efficiently (see, for instance, [81]). Solving a series of maximum-flow problems, one can derive the maximum network coding throughput. On the other-hand, computing the maximum multicast throughput boils down to consider the fractional Steiner tree packing problem, an NP-hard combinatorial problem [34, 47]. This problem can be related to another well-known NP-hard problem, namely the Steiner tree problem where one has to find a minimum cost subgraph interconnecting (or spanning) a given subset of nodes (called terminals). It is a common belief that these problems are almost impossible to solve in practice. However, using efficient Mixed-Integer Models

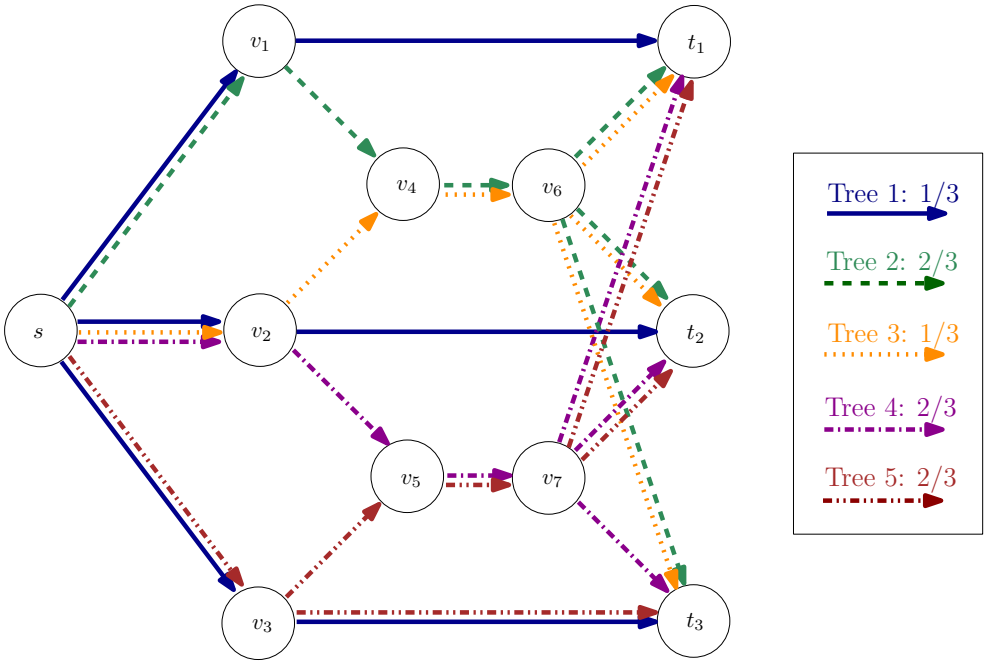


Figure 4.2: Optimal solution with Multicast (MC) in a butterfly network: throughput = 8/3, but 5 different multicast trees are required.

[37], large instances of Steiner tree problems can be solved in a few seconds [58]. As a result, significant instances of the fractional Steiner tree packing problem can also be solved and thus, maximum multicast throughput can be evaluated on large sets of instances. To the best of our knowledge, all the previous empirical approaches to compare network coding and multicast throughput have either used approximation algorithms to solve the Steiner tree packing problem, or have considered specific graphs instances where the resolution of the problem appeared to be more tractable. In [66], the authors consider the undirected case and claim that it is more general than the directed one. They also provide a Linear-Programming model using what they call *conceptual flows* to compute the network coding maximum throughput. Note that this model is nothing else than a very classical flow model that can trivially be derived from directed network coding models such as, for instance, the one in [69]. Moreover, the authors in [66] use a standard simplex algorithm to solve a problem for which very much more efficient combinatorial methods are available (max-flow algorithms). They restrict their attention to uniform bi-partite graphs, whereas real-life network topologies can have very different structures. Finally, they use a brute force algorithm to solve the Steiner tree packing problem by enumerating all possible trees and then claim that the problem cannot be solved even for medium size instances.

Since [66] seems to be the most comprehensive empirical study of network coding versus other routing paradigm throughput, we believe this chapter will fill some major gap in this important field. Our contributions are the following:

- We propose efficient resolution schemes for the multicast maximum throughput problem (fractional Steiner tree packing problem) in the directed, bidirected (for each link, there is one arc in each direction) and undirected graphs case. The chosen approach is a standard column generation algorithm where each column represents a Steiner tree. The pricing problems are hence Steiner tree problems for which we use a flow type model as in [37]. We also handle the maximum multicast throughput problem with a bounded number of trees, which, to the best of our knowledge, has not been addressed until now. For this problem, we propose an exact flow based model and a heuristic approach using tree variables.

- We somehow clarify the differences and relationships between directed, bidirected and undirected settings. We also claim that the bidirected case is quite relevant for telecommunication models because most fixed networks infrastructures often have two opposite links of same capacity.

- We provide numerical results based on extensive computational experiments (each set contains one hundred instances and the small topologies were checked with much larger number of instances). All instance are randomly generated without any predefined underlying structure and all results provided are the exact solutions of the related throughput problems.

Our main findings are:

- We confirm (but over a much larger scope of instances) what was announced in [66], namely that it is almost impossible to find instances exhibiting a non-zero gap between network coding and multicast throughput. In other words, the probability to pick a "butterfly-type" instance is almost zero.

- Concentrating on very small graphs (7 to 10 nodes), we have generated a very large number of instances and, in the smallest cases, even enumerated all possible instances. In this last setting, we were hence able to evaluate the probablity of picking up a topology with a non-zero gap, to be as low as 0.01%.

- Using again mixed-integer models, we could confirm numerically that the largest gap for uniform instances up to 7 nodes is 0.5.

- Finally, and this is not a surprise, we could evaluate the impact of the number of trees in the multicast solutions, and confirm that, when the number of trees is limited, there is often a huge gap between network coding and multicast throughput.

To summarize these results, we can say that, from a numerical point of view, it is wrong to claim that network coding allows to achieve a significant improvement in throughput. However, from a practical point of view, since telecommunication operators are reluctant, for obvious management complexity reasons, to handle large (and even moderate) sets of multicast trees, there is a very significant advantage that can be achieved by using network coding: network coding is much easier to deploy than multicast techniques, for a similar throughput when the number of trees to manage is large, and for a much better throughput when the number of trees is small. Of course, there are also many other interesting features offered by network coding (dynamicity, robustness, ...) which have been already largely promoted in the literature.

4.3 Comparing Coding and Routing Schemes

In this chapter, our aim is to compare the throughput achieved while using different coding/routing paradigms:

- **Network Coding (NC)**: we use the result of [5] to compute the throughput achieved using network coding and multicast forwarding. Solving independently the sequence of maximum flow problems allows to obtain the value of the network coding throughput. The result obtained for a small graph G_1 is depicted in Figure 4.1: the instance has one source, three terminals, all arcs have a capacity of 1, except the arc (s, v_2) of capacity 2. Three streams a, b and c (each one representing a volume of 1) are sent by the source node s. The nodes v_2, v_4 and v_5 perform coding on their input streams (for instance $a \oplus b$ means that streams a and b are coded together, resulting in a stream of volume 1). As a result, the terminal nodes receive each three different coded or uncoded streams, from which they can all decode the original streams a, b and c: the optimal network coding throughput is hence 3.

- **Multicast (MC)**: by Multicast, we mean here that several multicast trees can be set up between one source and all of its terminals, and the traffic is split appropriately among the chosen trees to reach the best possible

throughput. The optimal throughput in Figure 4.2 is $8/3 \approx 2.666$, using 5 trees (see Figure 4.2). The original data is hence sliced into 5 streams, each one is sent on a different multicast tree.

- **Multicast with ℓ Trees (MC-ℓ)**: since the previous case is not often realistic (because network operators will not agree to handle many different multicast trees rooted at the same source), we also consider the case where the streams can only be routed over a limited number of multicast trees (up to ℓ and, in the numerical experiments, we will only consider the cases $\ell = 1, 2$ or 3). The optimal throughput in Figure 4.2 is 1 with one tree and 2 with two trees.

4.4 Models and Algorithms

4.4.1 Notations

The network is modeled as a directed capacitated network $G = (V, A, C)$ where $C_a > 0$ is the capacity of arc $a \in A$, $n_V = |V|$, $n_A = |A|$. In the network, one source node s that have some data content may interest some other nodes $t^k \in T, T \subset V$ where $n_T = |K|$ and $k = 1, 2, \ldots, |K|$. we call the nodes in the set T terminals. The network will then set up a multicast session between s and T by using either multicast protocol or network coding.

We denote by $\mathcal{P}^k$ or $\mathcal{P}^{st^k}$ the set of simple paths (without cycles) between s and t^k, and by $\mathcal{P} = \bigcup_{k \in K} \mathcal{P}^k$ the set of all paths. Similarly, we denote by $\mathcal{T}(r, U)$ the set of trees rooted at $r \in V$ and spanning $U \subset V \setminus \{r\}$. If the root node or the set of spanned nodes is obvious from the context, they will be omitted in the notation. For any subset of nodes $W \subset V$, we denote by $\delta^+_G(W)$ and $\delta^-_G(W)$ the set of arcs leaving and entering W in the graph G. Again, the graph in subscript will be omitted if it is clear from the context. When the set W is reduced to a singleton $\{v\}$, we will use the notations $\delta^-(v)$ and $\delta^+(v)$ (instead of $\delta^-(\{v\})$ and $\delta^+(\{v\})$).

To model the throughput maximization problems, we will mainly use the following flow variables:

- f^k_a is the total flow sent from s to t^k on arc $a \in A$.
- f_a is the total flow on arc $a \in A$, $f_{(i,j)}$ is the total flow on arc $(i, j) \in A$.

- φ_p^k is the total flow sent on path $p \in \mathcal{P}^k$ (and simply φ_p for the total flow on path $p \in \mathcal{P}$).

- φ_τ is the total flow on the tree τ.

We now provide standard (LP=Linear Programming or MIP=Mixed Integer Programming) models for computing the maximum throughput according to the various routing paradigms.

4.4.2 Network Coding

As already mentioned in Chapter 2, network coding allows mixing of flows in the intermediate nodes when information is transmitted across a given network. The authors in [5] show that theoretically the novel data forwarding mechanism has significant benefit on data transmission, especially for multicast networks. They prove that network coding achieves the maximum end-to-end multicast throughput. The throughput is equivalent to the minimum value of the maximum flows between the source and each terminal in T. In order to find the multicast throughput when using network coding, we can compute the maximum flow between the source and each terminal successively.

Maximum Flow Problem

The maximum flow problem between two nodes in a directed graph is a sophisticated network flow problem that has already been extensively studied [26, 28]. The problem rose to fame during the second world war as a military application. It calculates the maximum amount of supplies that one city can get from another one through a rail network. After the development of telecommunication industry, it becomes useful in this field as well.

Many algorithms have been invented for solving this problem. The first well-known algorithm is proposed by Ford and Fulkerson in [28]. It starts by sending an arbitrary integer flow from source s to terminal t, of course, the capacity limitations on the path for sending this flow must be followed. In each iteration, the algorithm randomly searches a path that can send a positive flow from s to t in the so-called residual graph. It stops when there is no such path in the residual graph. Given a directed graph $G = (V, A)$ and a flow f_p that is sent on a path p in G, the residual graph $G^f = (V^f, A^f)$ is defined as follows:

- $V^f = V$;

- $\forall (i,j) \in p$, $C^f_{(i,j)} = C_{(i,j)} - f_p$;

- $\forall (i,j) \in p$, create arc (j,i) with capacity f_p, if there is no such an arc, or $C^f_{(j,i)} = C^f_{(j,i)} + f_p$;

A path that is able to carry a positive flow from s to t in the residual graph is called augmenting flow in the literature. The algorithm works merely if all the capacities are integers, otherwise it will not converge to the maximum value. A more robust algorithm called Edmonds-Karp algorithm [26] has been proposed by J. Edmonds and R. Karp. It is a special version for implementing Ford-Fulkerson algorithm in polynomial time $O(VA^2)$. It specifies the way for searching an augmenting path, which applies a breadth-first search to find a shortest path that has available capacity.

This problem can also be formulated by using mathematical programming skills. Given a directed graph G, we set an variable R which is denoted by an amount of flow R to the terminal t and needs to be maximized. The mathematical model for the maximum flow problem is given as follows:

$$\begin{aligned}
\text{maximize} \quad & R, && && (4.1)\\
\text{s.t.} \quad & C_a \geq f_a, && \forall a \in A, && (4.2)\\
& \sum_{a \in \delta^-(v)} f_a - \sum_{a \in \delta^+(v)} f_a = b_v(R), && \forall v \in V, && (4.3)\\
& f_a \geq 0, && \forall a \in A, && (4.4)
\end{aligned}$$

where the constants b_v are defined as:

$$b_v(x) = \begin{cases} -R & \text{if } v = s \\ R & \text{if } v = t \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

In the formulations from (4.1) to (4.4), The constraints (4.2) are capacity constraints, and the constraints (4.3) are flow conservation constraints. This is an arc-based formulation. Normally, the arc-based formulation can be interchanged with the path-based formulation which means that we use path flow variables φ_p rather than flow variables f_a. The path-based formulation is given in (4.6)-(4.8).

We denote the value of the maximum flow between s and t by $\varphi^*(s,t)$:

$$\text{maximize} \quad \varphi^*(s,t) = \sum_{p \in \mathcal{P}^{st}} \varphi_p, \tag{4.6}$$

$$\text{s.t.} \quad \sum_{\substack{p \in \mathcal{P}^{st}: \\ a \in p}} \varphi_p \leq c_a, \qquad \forall a \in A, \tag{4.7}$$

$$\varphi_p \geq 0, \qquad \forall p \in \mathcal{P}^{st}, \tag{4.8}$$

The constraints (4.7) are capacity constraints, and the objective function is to maximize the sum of the flows that are sent on all the paths between s and t in a given network.

From the mathematical programming point of view, it is more practical to use the path-based formulation with a so-called column generation method, which we will introduce in the next section, than the arc-based one especially in large scale networks. The reason is that, during the process of implementation, the storage used by arc-based formulation is normally larger than the path-based one. For example, assume that a graph has 1000 arcs and 1000 nodes, then from the arc-based formulation, we see that there are 1000 variables and 2000 constraints. On the contrary, there are only 1000 constraints in the path-based constraints. Moreover, although the paths' variables in the path-based formulation are much more than the ones in the arc-based formulation, column generation method merely uses a small portion of the paths' variables, which normally requires less time for solving the same problem.

Column Generation Method

Column generation is a method that allows solving efficiently the LP problems with a large number of variables. To the best of our knowledge, the original idea of column generation is proposed by Ford and Fulkerson for solving multi-commodity flow problem in [30]. We assume that a LP, which is called a *master* LP, is to be solved, then the column generation method starts from a restricted master LP which solves a manageable problem with the same constraint sets but only a small set of variables of the original problem. By analyzing the partial solution, we see whether the solution can still be improved. If the solution is not optimal, we then introduce a column variable in the previous restricted master LP to enlarge the model, and the new model will be resolved again. Column generation repeats the process until the problem achieves the optimal solution.

The analysis on whether an additional column of variables needs to be added and which variable is to be introduced is based on a subproblem called *pricing problem* which evaluates the reduced cost that is associated with each variable. The reduced cost measures how much the objective function must be improved before a variable becomes positive in the solution set. In the maximization problem and in each iteration of column generation, we find the best positive reduced cost that can increase the most in the objective function. When there is no positive reduced cost, the column generation finds the optimal solution.

Returning to the path formulation of the maximum flow problem, we can derive easily the reduced cost for each path variable:

$$r_p = 1 - \sum_{a \in p} w_a \tag{4.9}$$

where w_a denotes the dual variable for each constraint in (4.7) and can be obtained by solving the restricted master LP. From (4.9), we know that finding the maximum r_p is equivalent to finding the minimum $\sum_{a \in p} w_a$ that is indeed a shortest path problem for a given graph with arbitrary weights given by corresponding dual variables.

Algorithm 4.4.2.1 : Column Generation Method for Maximum Flow Problem

1. Choose one arbitrary path variable to create a restricted master of maximum flow problem (r-MF);
2. **repeat**
3. Solve r-MF and let $r_a, a \in A$ be the dual variables of r-MF;
4. Find the shortest path between s and t in $G = (V, A)$ with weights $r_a, a \in A$;
5. **if** The weight of the shortest path is less than 1 **then**
6. Add φ_p to r-MF;
7. **else**
8. The optimal solution has been found; exit the algorithm;
9. **end if**
10. **until** No path has been added

Max-Flow Min-Cut Theorem

Similar to the cut definition we give in Chapter 3 Sec. 3.3.1, a $s-t$ cut in graph $G = (V, A)$ is a node partition $E(S, T)$ where s is in S and t is in T and where $S \subset V$, $T \subset V$, and $S \cap T = \emptyset$. A minimum cut problem is a problem which

finds an $s-t$ cut with minimum sum capacities among all the possible $E(S,T)$. Surprisingly, the problem is the dual problem of the maximum flow problem. The algorithm proposed by Ford and Fulkerson implicitly proves the strong duality of the maximum flow problem and shows the max-flow min-cut property in [29], which tells that the value of the maximum flow in a given graph is equivalent to the minimum sum capacities of a set of edges that can separate s and t.

In 2000, Alshswede *et al* in [5] extend the max-flow min-cut concept to network coding flow in telecommunications. In their paper, they characterize the maximum possible flow that can be multicast from source s to several terminals $T \subset V$. The maximum flow is equivalent to the minimum value of all the maximum flows between s and each terminal $t \in T$.

Network Coding Flow

According to the previous section, the maximum NC throughput is then given by:

$$NC : \varphi^*(s) = \min_{t \in T} \varphi^*(s,t). \tag{4.10}$$

Although the computation of $\varphi^*(s)$ can be done by applying iteratively the corresponding max flow algorithm between s and each t^k, the problem can also be captured within a linear programming flow model:

$$NC_{base} \begin{cases} \max \lambda^{nc}, & & (4.11) \\ \displaystyle\sum_{p \in \mathcal{P}^k} \varphi_p = \lambda^{nc}, & \forall k \in K, & (4.12) \\ \displaystyle f_a \geq \sum_{\substack{p \in \mathcal{P}^k: \\ a \in p}} \varphi_p, & \forall a \in A, k \in K, & (4.13) \\ f_a \leq C_a, & \forall a \in A, & (4.14) \\ \varphi_p, f_a \geq 0, & \forall a \in A, p \in \mathcal{P}^k. & (4.15) \end{cases}$$

In this model, λ^{nc} is the unknown throughput value that is to be maximized, constraints (4.12) certify that λ^{nc} units of flow are indeed sent towards each terminal (potentially split over several paths), constraints (4.13) are used to compute the resulting flow on each arc and constraints (4.14) are the so-called capacity constraints. Note that the flow variables f_a are not really needed in this model (they could be eliminated by aggregating constraints (4.13) and (4.14)). However, it is interesting to keep these variables because they give an indication on the amount of bandwidth that will be used on each arc. Note that constraints (4.13) and

(4.14) imply:

$$\max_{k \in K} f_a^k \leq f_a \leq C_a, \ \forall a \in A, \tag{4.16}$$

where f_a^k is used, as previously, to denote the right-hand-side of constraints (4.13). Note that, in an optimal solution of our linear programming model, each flow variable f_a can take any value within the bounds specified by (4.16). Since we are interested in recovering actual flow values, we will assume that $f_a = \max_{k \in K} f_a^k$, for all $a \in A$ (this can easily be achieved by post-optimization or even by considering an auxiliary problem).

4.4.3 Multicast Routing on Multiple Trees

We already study how to find the maximum throughput when using a single multicast tree in Chapter 3. But in fact, in a dense graph, the multicast throughput can still be improved by using several different multicast trees. In this section, we study how to model multicast routing with multiple trees. There are several ways to model this problem. We provide here a simple model based on the assumption that we are able to generate efficiently trees spanning T (the so-called *Steiner trees*):

$$MC_1 \begin{cases} \max \sum_{\tau \in \mathcal{T}} \varphi_\tau, & (4.17) \\ \sum_{\substack{\tau \in \mathcal{T} \\ a \in \tau}} \varphi_\tau \leq c_a, \qquad \forall a \in A, & (4.18) \\ \varphi_\tau \geq 0, \qquad \forall \tau \in \mathcal{T}. & (4.19) \end{cases}$$

Again, in a given graph, there is a large number of trees which indicates that there are a large amount of corresponding tree variables. Having the knowledge from Column generation method, we first derive the reduced cost for a tree variable. If we denote by w_a the dual variable associated with the capacity constraints (4.18), then the reduced cost of tree τ is:

$$\overline{r}_\tau = 1 - \sum_{a \in \tau} w_a. \tag{4.20}$$

The pricing problem then consists in finding a minimum weight tree (in the graph weighted by the dual variables w_a) rooted at s and spanning the subset of terminals $T \subset V$. When $\{s\} \cup T \neq V$, this problem turns out to be a Steiner tree

problem which can be sumarized as follows:

$$ST(w) : \min_{\tau \in \mathcal{T}(T)} \sum_{a \in \tau} w_a. \tag{4.21}$$

The formulations of this subproblem is the same as the ones shown in (3.15)−(3.18). Although the subproblem is NP-hard, but almost all the instance we generated can still be solved very efficiently, in terms of time consuming, by using commercial solver, for example, Xpress Optimizer. The problem (MC_1) is nothing else than a *Fractional Steiner Tree Packing* problem [47] which is also NP-hard. When we use column generation in a medium or large size network, the algorithm is very efficient, we will confirm this argument in our numerical test section.

Since the number of trees used in a multicast solution is a main concern (because it impacts greatly whether solutions can be implemented in practice), we also considered a second phase algorithm where, the optimal throughput value being fixed, we minimize the number of trees used (which might be smaller than the number of trees used in the optimal solution).

4.4.4 Multicast Routing on a Limited Number of Trees

Limiting the number of trees in the multicast model adds a significant difficulty to the model. Indeed, it is necessary to count the trees used to carry some traffic, so additional binary variables must be introduced into the models:

$$MC_1^\ell \begin{cases} \max \sum_{\tau \in \mathcal{T}} \varphi_\tau, & & (4.22) \\ \sum_{\substack{\tau \in \mathcal{T} \\ a \in t}} \varphi_\tau \leq C_a, & \forall a \in A, & (4.23) \\ \varphi_\tau \leq C x_\tau, & \forall \tau \in \mathcal{T}, & (4.24) \\ \sum_{\tau \in \mathcal{T}} x_\tau \leq \ell, & & (4.25) \\ \varphi_\tau \geq 0, x_\tau \in \{0,1\} & \forall \tau \in \mathcal{T}. & (4.26) \end{cases}$$

Here, the constraint (4.25) limits the number of trees used in a solution to a maximum of ℓ. The constant C can be set to the maximum of all capacities. Constraints (4.24) are used to set to 0 the flows on trees when the associated binary variable is equal to 0.

Since it is intractable in practice to consider explicitly all possible trees of $\mathcal{T}$, to

solve (MC_1^ℓ), one must again consider column generation phases where a pricing sub-problem is solved to find candidate trees to add into the master problem (as in Section 4.4.3). However, since the master problem is a MIP, its resolution involves branching phases and the pricing subproblem must be solved in each node of the branch-and-bound tree (branch-and-price). Moreover, the pricing problem has to be adapted to fit each local version of the master problem.

One way to avoid this tedious resolution process consists in relying on simple heuristics where a limited set of candidate trees $\hat{\mathcal{T}} \subset \mathcal{T}$ is generated in a first phase (for instance by considering various perturbed versions of the initial master problem) and then the MIP is solved once with this unique set $\hat{\mathcal{T}}$:

Algorithm 4.4.4.1 : Simple heuristic for MC_1^ℓ

Input: directed capacitated graph $G = (V, A, C)$, set of single-source streams $\{s, t^k, d^k\}_{k \in K}$;

Output: a lower-bound on the multicast maximum throughput using fewer than ℓ trees;

1. Solve Problem MC_1;
2. If $|\mathcal{T}^{MC}| \leq \ell$, then the continuous solution of MC is also optimal for MC_1^ℓ: stop the algorithm;
3. Otherwise, let $\hat{\mathcal{T}} \leftarrow \mathcal{T}^{MC}$;
4. Solve problem MC_1^ℓ defined over the restricted set of trees $\hat{\mathcal{T}}$;

Note that, when ℓ, the number of trees, is small, we have also considered an alternate exact formulations for (MC_1^ℓ).

4.5 Numerical Experiments

The previous section provides guidelines to solve the considered maximum throughput problems. In this section, we use a commercial tool (XPress Optimizer Version 21.01.00) to solve LP (Linear Programming) and MIP (mixed Integer Programming) problems.

We have conducted several computational experiments on directed and on bidirected euclidean instances. These instances were randomly generated using Algorithm 4.5.0.2.

Series of 100, 1000 or even 10,000 random instances have been generated for several sets (n, m, k_s, k_t). Note that for bidirected instances, m represents the number of links, so that the number of arcs is $2m$. For directed instances, for

Algorithm 4.5.0.2 : Generate Directed/Bidirected Instance

Input: number of nodes n, number of arcs m, number of sources k_s, number of terminals k_t, a geographical box $\Omega = [a_1, b_1] \times [a_2, b_2]$, a capacity interval $[C_{min}, C_{max}]$;

Output: a connected capacitated undirected graph $G = (V, E, C)$ within Ω;

1. Generate uniformly n nodes in $\Omega \rightarrow V$;
2. Compute a minimum distance spanning arborescence/tree in the complete graph K_n;
3. Randomly add $m - n + 1$ arcs/edges with a probability inversely proportional to the distance between the end-nodes;
4. Randomly generate arc/edge capacities in $[C_{min}, C_{max}]$;

each edge $\{i, j\}$ to generate, we decide randomly to introduce arc (i, j), arc (j, i) or both at a time. We have applied our maximum throughput algorithms on all these instances and average results are reported.

The first surprising result is that, over all the generated instances, directed and bidirected, we could not find a single one where network coding (NC) had a larger maximum throughput than multicast (MC). However, when we considered multicast with a limited number of trees (MC-ℓ), the situation was quite different. Figure 4.3 shows the relative throughput values (100% means NC and MC throughput) achieved when restricting multicast to use only 1, 2 or 3 trees. We see that the throughput value decreases as the number of trees decreases. The general trends are very similar for the directed and bidirected instances. The reduction of throughput is much more important when the instances are denser (about $6n$ edges for the first instance set and $3n$ edges for the other). This is due to the fact that a limited number of trees cannot exploit the full potential offered by the network whereas Network Coding does. In traditional telecommunication networks, the average degree is usually rather low (say between 3 and 5). The observations made on the four last series of instances show that there is still a significant throughput reduction (from 13 to 25 %) when using up to 3 multicast trees, when compared to a network coding solution. Network coding should hence be considered by network administrators as a very attractive alternative solution to standard routing solutions.

Figure 4.4 gives some insight on the number of trees required in a multicast setting to reach the same throughput as if NC was used. The resulting numbers fall within the boxes for 50 % of the generated instances (and within the intervals for 90 % of the instances). We can observe that the variance in the optimal num-

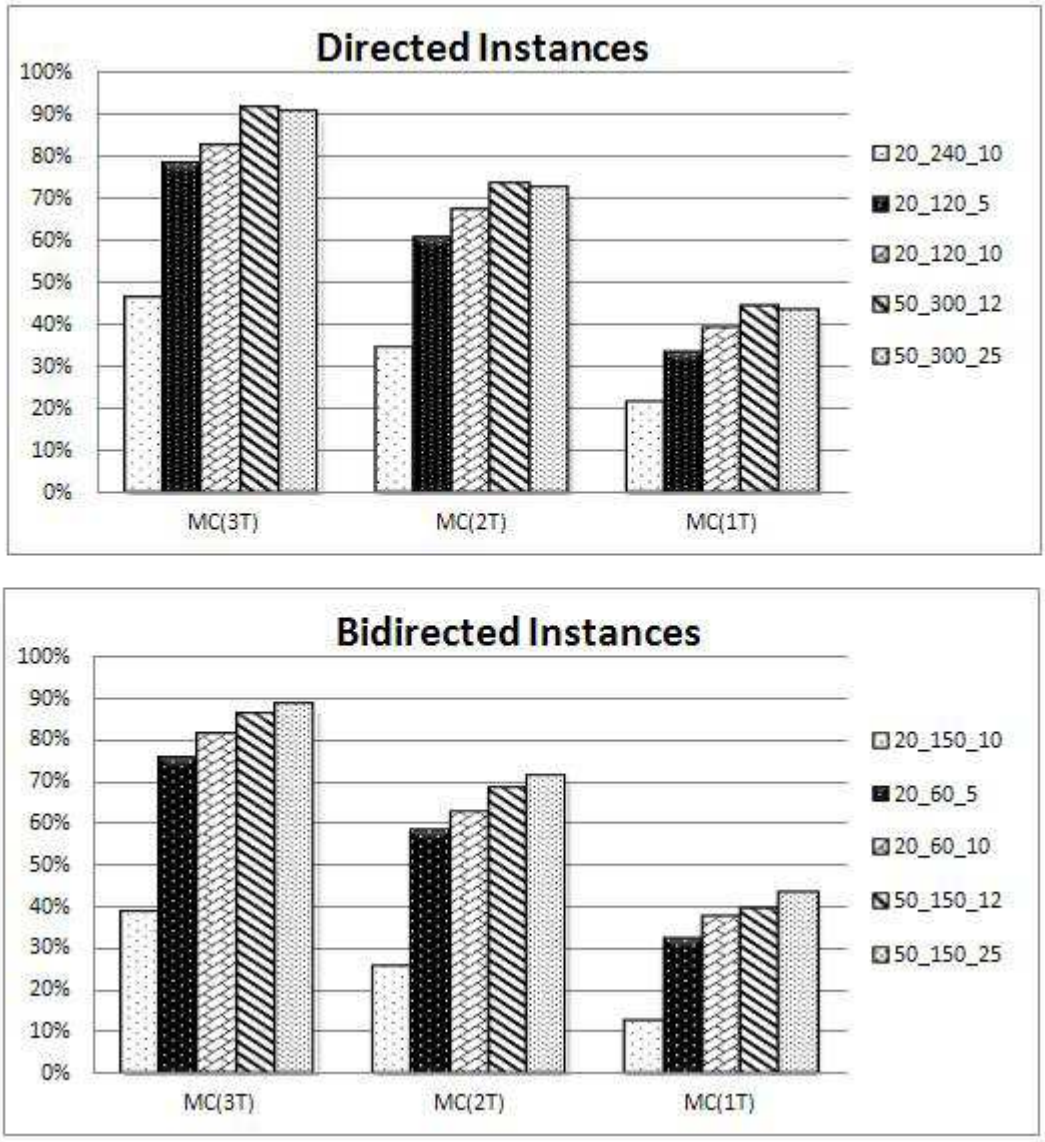


Figure 4.3: Throughput comparison between multicast and network coding: (Note that the legend, for example 20_240_10, denotes a random generated graph that has 20 nodes, 240 arcs, and 10 out of 20 nodes are terminals. This type of legend will be used through the rest of this chapter.) Relative throughput achieved for MC with 1, 2 and 3 trees over different sets of instances (given by n, m and kt, all with a single source: $ks = 1$). The values provided are the average over 100 random instances of each type. The maximum (100 %) corresponds to the NC and unrestricted MC throughputs.

ber of trees is quite high, some instances requiring relatively few trees whereas others require a large number of trees. The ranges and values decrease when the number of terminal increases. This can be related to the fact that, thanks to Edmonds arborescence packing theorem [25], the theoretical gain of Network

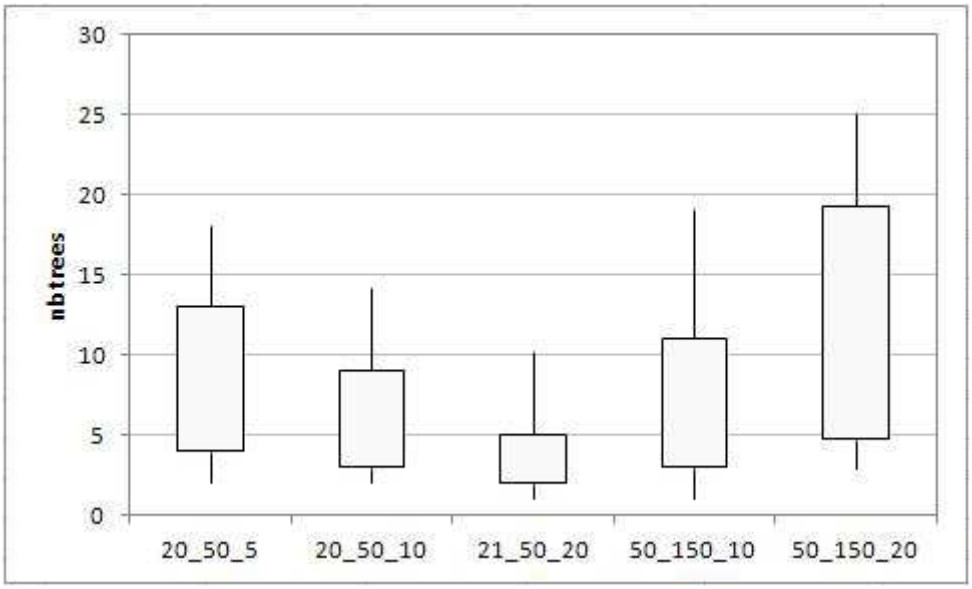


Figure 4.4: Statistical result of the number of trees for achieving optimal throughput: for each set (indicated on the x-axis), 100 random instances are generated and the minimal number of multicast trees required to obtain the optimal throughput is computed. The resulting values for 50% of the instances fall within the boxes (and for 90 % within the intervals).

Coding throughput vanishes when ALL nodes are terminal. If five trees would be considered as a reasonable upper-limit for operators to handle, then, in most cases, the NC throughput would not been achieved with multicast. On the other hand, ten trees would often suffice for small networks.

In Figure 4.5, we provide the average, over all generated instances of a given series, of the computing times for obtaining the optimal throughput values using our models. We clearly see that multicast throughput computations generally take a few minutes whereas network coding throughput computation is instantaneous. Computations for a single multicast tree are also very fast, but for values larger than 2, the problem becomes increasingly long to solve. We have performed many other computational experiments, including some with multiple source instances, and the main observations remain generally the same.

4.6 Considerations on Some Small Instances

The main outcome of our first numerical experiments is that Network Coding (NC) never seems to outperform the throughput allowed by Multicast solutions (MC). Focusing more on small size instances, the results in Table 4.1 show that all

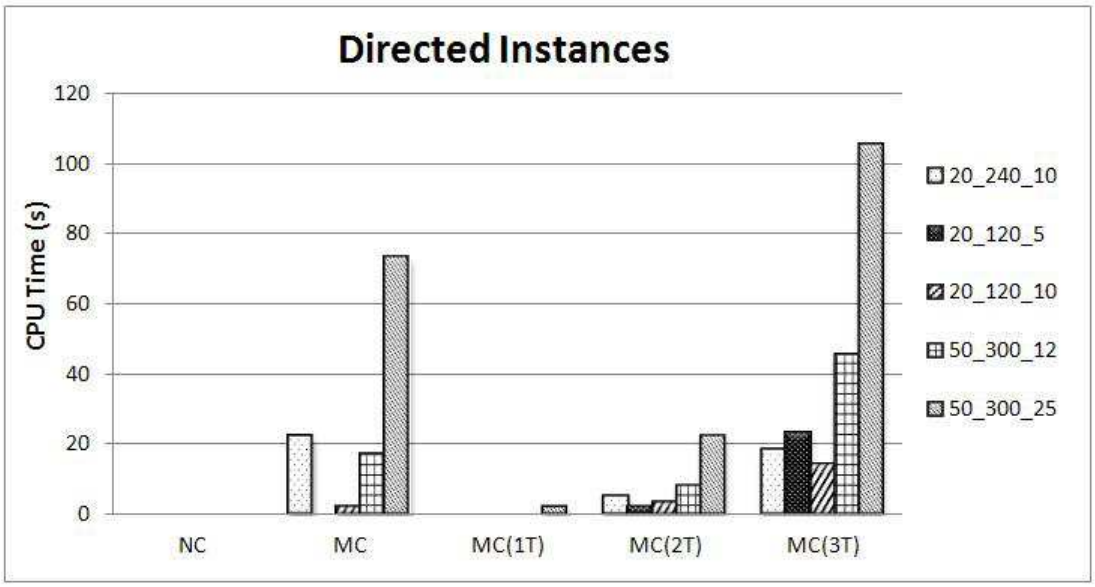


Figure 4.5: Average computing times (in seconds) for solving the various maximum throughput problems.

uniform instances (all capacities are equal to 1) generated with 7 to 10 nodes have the same maximum throughput with NC and MC. In particular, this means that our generator was not able to reproduce the "classical" butterfly network. To get some insight on this surprising phenomenon, we decided to perform an exhaustive search for the restricted case of graph with 7 nodes, 1 source, 2 terminals and all capacities equal to 1. To limit somewhat the search, we have only considered instances with at least two outgoing arcs from the source and with at least two incoming arcs into the terminals. Among the 950,951 possible graphs (removing 18,016 disconnected cases), only 96 showed a non-zero gap between NC and MC. In fact, these 96 cases can be essentially reduced to the 3 graphs displayed in Figure 4.6, all other cases being symmetrically equivalent to those 3. Observe that the first one is the classical butterfly graph, whereas the two other are small variants. If we consider a uniform distribution among all graphs, then the probability to pick a graph with a non-zero gap is about 0.01%.

One step further in this line of research consists, instead of enumerating all possible instances of a certain type, in considering the problem of finding the maximum gap as an optimization problem. For this purpose, we define binary design variables $x_a \in \{0, 1\}$ which will be set to 1 for the arcs that are kept in the final topology (assuming we start from a potentially fully meshed topology, i.e., a

Set	n	m	ks	kt	λ_{NC}	λ_{MC}	# trees
Instances n_rand_rand							
AVG	7	14.35	1	4.03	2.95	2.95	4.83
MIN	7	8	1	2	1	1	1
MAX	7	21	1	6	6	6	30
AVG	8	18.24	1	4.52	3.25	3.25	6.15
MIN	8	9	1	2	1	1	1
MAX	8	28	1	7	7	7	42
AVG	9	22.71	1	5	3.54	3.54	7.3
MIN	9	10	1	2	1	1	1
MAX	9	36	1	8	8	8	56
AVG	10	27.81	1	5.43	3.9	3.9	9.08
MIN	10	11	1	2	1	1	1
MAX	10	45	1	9	9	9	72

Table 4.1: Statistical results in random graphs with uniform capacities:(average over 1000 instances) results for the single-source case with uniform capacity $C_a = 1, \forall a$.

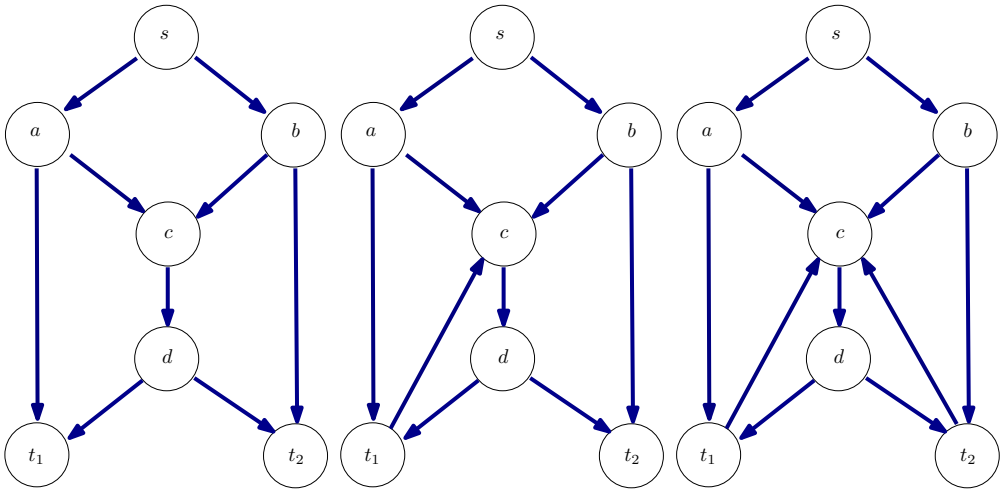


Figure 4.6: The only 3 graphs with a non-zero gap (of 0.5) between NC and MC throughput (in the case of uniform capacity $C_a = 1, \forall a$).

complete graph). We will also assume that the number of nodes n, the number of streams m are given, as well as candidate capacities C_a, one for each arc. Here, we will mainly consider uniform capacity models ($C_a = 1, \forall a \in A$) and single source models. Hence, in the instances, kt will specify the number of terminals. So, for instance, the setting ($n = 10, kt = 5$) concerns all instances with 10 nodes, one source node and 5 terminals. The optimization problem then consists in finding the subgraph of K_n with kt terminals, where the gap between NC and MC

throughput is maximal. So basically, the problem could be summarized as:

$$\Delta^*(NC, MC) = \max_{x_a \in \{0,1\}} \left(\lambda^*_{NC} - \lambda^*_{MC}\right), \tag{4.27}$$

where λ^*_{NC} and λ^*_{MC} are the maximum throughput achieved by NC and MC, respectively, and are thus, the results from their own maximization problems. For the NC throughput maximization problem, we are going to rely on arc-flow variables (instead of flow-path variables as used up to now): $\lambda^*_{NC}(\mathbf{x}) =$

$$\lambda^*_{NC}(\mathbf{x}) = \begin{cases} \max \lambda, & & (4.28) \\ \displaystyle\sum_{a \in \delta^-(v)} r^k_a - \sum_{a \in \delta^+(v)} r^k_a = b^k_v(\lambda), & \forall v, k, & (4.29) \\ r^k_a \leq c_a x_a & \forall a, k, & (4.30) \\ r^k_a \geq 0, & \forall a, k. & (4.31) \end{cases}$$

Since we need to substract MC maximum throughput in our global problem, we choose to express λ^*_{MC} in a LP-dual form:

$$\lambda^*_{MC}(\mathbf{x}) = \begin{cases} \min \displaystyle\sum_{a \in A} c_a x_a v_a, & & (4.32) \\ \displaystyle\sum_{k \in K} (\pi^k_s - \pi^k_{t^k}) = 1, & & (4.33) \\ u^k_{ij} \geq \pi^k_i - \pi^k_j, & \forall i, j, k, & (4.34) \\ v_a \geq \displaystyle\sum_{k \in K} u^k_a, & \forall a, k, & (4.35) \\ \pi^k_i, u^k_a, v_a \geq 0, & \forall a, i, k. & (4.36) \end{cases}$$

This is the standard dual formulation of an equivalent arc-flow formulation of MC_1 (eq. (4.17)-(4.19)). Note that, when we consider both problems jointly in (4.27), the objective of the MC throughput optimization subproblem becomes non-linear (product of variables x_a and v_a). Using standard linearization techniques, we can obtain a single MIP formulation for the joint problem (4.27). As it is often the case in such linearized problems, the continuous relaxation bound is very weak. To further tighten the model, we have also used series of symmetry breaking constraints, in order to avoid to consider multiple similar solutions.

Plugging this model in the Xpress solver, we were able to solve a few small size instances (see Table 4.2). It is interesting to note that, for uniform instances with 7 and 8 nodes, the only cases with a non-zero gap between NC and MC maximum throughput are the ones having 7 nodes with 2 destinations, 8 nodes with 2 or

n	k	c_a	$\Delta^*(NC, MC)$	λ^*_{NC}	λ^*_{MC}	m	cpu (sec)
7	2	1	0.5	2	1.5	9	0.8
7	3	1	0	-	-	-	0.2
7	4	1	0	-	-	-	0.1
7	5	1	0	-	-	-	0.1
7	6	1	0	-	-	-	0.1
8	2	1	0.5	3	2.5	13	380
8	3	1	0.5	2	1.5	11	12
8	4	1	0	-	-	-	0,5
8	5	1	0	-	-	-	0,1
8	6	1	0	-	-	-	0,1
8	7	1	0	-	-	-	0,1

Table 4.2: Results for computation of maximum throughput gap between NC and MC (uniform instances).

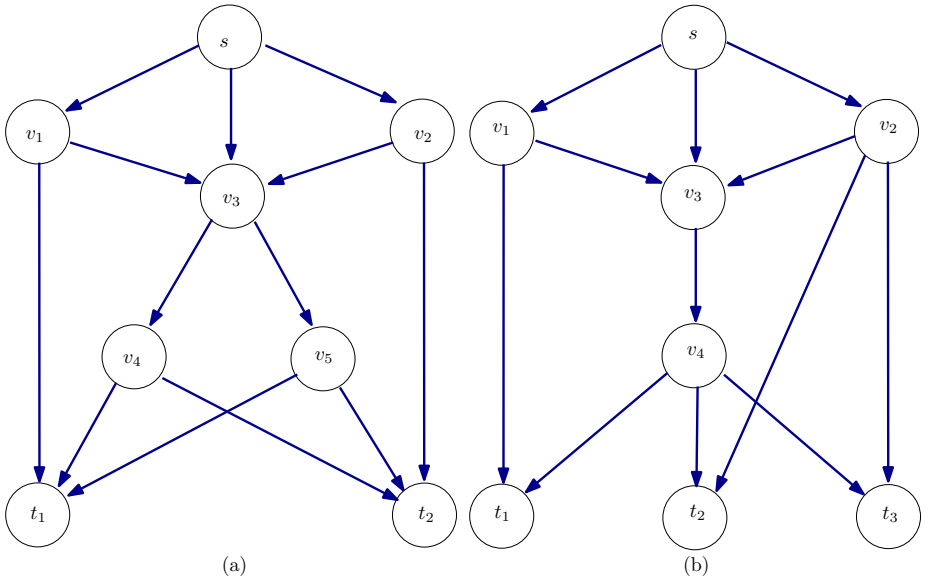


Figure 4.7: Two 8 nodes graphs whit a 0.5 gap between uniform NC and MC throughput maximum flows.

3 destinations. The first case corresponds to the classical "butterfly network". Two graphs for the 8 nodes, 2 or 3 terminals case are displayed in Figure 4.7. It is easy to check that, in these two graphs, the gap is indeed 0.5.

Note that we have used here an experimental way of establishing the following result:

Lemma 4.6.1. *Consider directed networks with n nodes ($n = 7$ or 8), unit capacities $c_a = 1, \forall a \in A$), a single source and k terminals (different from the*

source node). If $n = 7$ and $k \geq 3$ or $n = 8$ and $k \geq 4$, then network coding does not improve the throughput with respect to standard multicast.

4.7 Conclusion

In this chapter, we have used algorithms mainly based on LP and MIP models, to solve large sets of throughput maximization problems, using either Multicast routing (MC) alone, or combined with Network Coding (NC) techniques. Our main finding is that situations where network coding improves the throughput almost never occur, and is only obtained in very specific instances, and this observation holds for directed and bi-directed settings. However, when we consider routing scenarios where the number of multicast trees is limited, network coding clearly allows a very significant improvement in throughput, apart from all other advantages of network coding that have already been largely advertised in the literature.

Chapter 5

Strategic Network Coding

Pure mathematics is, in its way, the poetry of logical ideas.

-Albert Einstein (1879-1955)

5.1 Summary

We consider the problem of introducing network coding in a network in a manner that balances the benefits obtained from coding with the costs of providing coding, where such costs depend both on the number of nodes performing coding and the volume of traffic that is coded. Previous work [53, 55] and [68] has envisaged either the minimization of the number of nodes performing coding, which is a NP-hard problem, or the trade-off between the volume of the coded traffic and the throughput gains obtained from coding. We provide an optimization framework that considers both parameters jointly and also considers the use of duplication versus coding at nodes. Traffic is multicast in a hybrid way that combines a tree with a coded multicast graph. Our results indicate that the gains of coding, which increase markedly with graph density, can be obtained with very few nodes' performing coding, but that most, though by no means all, traffic in a multicast session should be coded for the bulk of the throughput gains to be achieved for that session.

5.2 Introduction

According to the observation from the last chapter, we know that network coding outperforms, in terms of throughput gain, multiple multicast routing strategy, in terms of throughput gain, when the number of multicast trees is limited. In this chapter, we are interested in evaluating the benefit of introducing limited network coding within a multicast meshed network. By "limited", we intend to capture the practical issue of a network operator that would like to deploy some network coding without disrupting too much the existing architecture and routing schemes, and remain with a limited budget. To deploy network coding, one needs to install coding and decoding software within the routing equipments (or nodes). A first way to control the network coding budget consists in limiting the number of nodes where coding is actually deployed. Moreover, one could also consider limiting the number of nodes where decoding is activated. But in a multicast setting where several (potentially many) users require the same content, offering the benefit of coding/decoding only to a subset of users could seem unfair. Alternatively, it could be an incentive for users to take in charge a part of the decoding cost. If we assume that the network operator wishes to offer the same service to all its clients, then the decoding features should be installed on every access node. In this case, another way to limit the cost incurred by decoding is to apply the decoding only to a small fraction of the overall traffic. This way, the decoding software would have to treat smaller volumes of data and would hence require less CPU. Going further, we could hence also argue that this would limit the energy consumption on the decoding equipments.

The problem of evaluating the cost of a network coding solution is not new. Several papers deal with cost evaluation and cost minimization of network coding schemes within multicast networks [69, 70, 71]. In these papers, the cost criterion is the cost of sending traffic over links (link cost). In [68], the author already considered a combination of multicast and network coding, but the multicast setting is based on a tree-packing model and does not consider solutions with small number of multicast trees. In [53, 55], a Genetic Algorithm is used to find the minimum number of coding nodes required to achieve a given throughput value. It is shown that the related problem is NP-hard and that, most of the times, very few coding nodes are sufficient to provide the maximum throughput benefit. In [17], a greedy algorithm is proposed to locate a set of network coding nodes in order to minimize an estimated decoding delay.

In this chapter, we propose to evaluate various scenarios where limited network coding is introduced in an existing (or together with a) multicast meshed network. It is well-known that a general multicast network can achieve almost the same throughput as a network coding solution [66, 67, 86]. However, these optimal multicast solutions often involve a very large number of trees, which makes them impractical. Conversely, when only a small number of multicast trees is used, the throughput can be very small as compared to network coding throughput [39]. This is why, in this chapter, we focus on routing schemes based only on one or two multicast trees, which seems reasonable to manage for a same service. Within this setting, we will consider additional, but limited, network coding features aiming at improving the overall throughput.

We compute the various throughput values using generic flow models (a flow in a graph modeling a stream) formulated as linear or mixed-integer programming problems. Some of these models are already well-known, but the novelty lies in the way we combine them in order to capture the various considered scenarios. Most of the problems we consider are NP-hard (for instance, the Steiner tree problem to design a single multicast tree, or the fractional Steiner tree packing problem to evaluate the throughput using several multicast trees), but using efficient models together with modern solving tools, we were able to obtain optimal solutions for all considered instances.

The contribution of this chapter is twofold: the various mixed-integer programming models we propose to compute the maximum throughput in several mixed scenarios combining network coding and multicast routing are, to the best of our knowledge, new, or have not been used in such a context. The second contribution is the findings we obtain when using these models: we found out that network coding can be introduced progressively in an existing multicast network (based on a limited number of multicast trees) while immediately producing a benefit in terms of throughput. To gain the full network coding benefit, a significant share of the traffic must be treated with network coding. We also observe that a few coding nodes suffice to produce the full throughput gain.

This chapter is organized as follows: in Section 5.3, classical flow models for computing maximum throughput, either using network coding or a single multicast tree, are reminded. Section 5.4 provides our assumption regarding the way coding or multicast replication functionalities can be modeled by observing the flow traversing a given node. In Section 5.5, we describe our extended models allowing

to control the number of coding/multicast nodes and to combine network coded traffic with the traffic sent on a single multicast tree. In Section 5.6, we describe in detail the results obtained on three butterfly-like instances. More numerical results based on randomly generated instances are proposed and analyzed in Section 5.7. Finally, in Section 5.8, we give some arguments to justify our models and show that they are quite efficient in practice.

5.3 Classical Flow Models

5.3.1 Notations

In this chapter we will use some notations that have already been defined in Chapter 4 Section 4.4.1, and we reproduce them here for convenience.

The network is modeled as a directed capacitated network $G = (V, A, C)$ where $C_a > 0$ is the capacity of arc $a \in A$, $n_V = |V|$, $n_A = |A|$. In the network, one source node s that have some data content may interest some other nodes $t^k \in T, T \subset V$ where $n_T = |K|$ and $k = 1, 2, \ldots, |K|$. We call the nodes in the set T terminals. The network sets up a multicast session between s and T by using either multicast protocol or network coding.

We denote by $\mathcal{P}^k$ or $\mathcal{P}^{st^k}$ the set of simple paths (without cycles) between s and t^k, and by $\mathcal{P} = \bigcup_{k \in K} \mathcal{P}^k$ the set of all paths. Similarly, we denote by $\mathcal{T}(r, U)$ the set of trees rooted at $r \in V$ and spanning $U \subset V \setminus \{r\}$. If the root node or the set of spanned nodes is obvious from the context, they will be omitted in the notation. For any subset of nodes $W \subset V$, we denote by $\delta^+_G(W)$ and $\delta^-_G(W)$ the set of arcs leaving and entering W in the graph G, respectively. Again, the graph in subscript will be omitted if it is clear from the context. When the set W is reduced to a singleton $\{v\}$, we will use the notations $\delta^-(v)$ and $\delta^+(v)$ (instead of $\delta^-(\{v\})$ and $\delta^+(\{v\})$).

Moreover, we denote by S the set of nodes that are neither sources nor terminals (sometimes called *Steiner nodes*). Considering any subset of arcs $Q \in A$, and a vector $(x_a)_{a \in A}$, we will often use the notation $x(Q)$ instead of $\sum_{a \in Q} x_a$.

To model the throughput maximization problems, we will mainly use the following flow variables:

- f_a^k is the total flow sent from s to t^k on arc $a \in A$.
- f_a is the total flow on arc $a \in A$, $f_{(i,j)}$ is the total flow on arc $(i,j) \in A$.
- φ_p^k is the total flow sent on path $p \in \mathcal{P}^k$ (and simply φ_p for the total flow on path $p \in \mathcal{P}$).
- φ_τ is the total flow on the tree τ.

5.3.2 Network Coding Model

The network coding model has been given in Chapter 4, Section 4.4.2, with very explicit explanation, in order to keep the independence for each chapter, we reprint the model here again.

$$NC_{base} \begin{cases} \max \lambda^{nc}, & (5.1) \\ \sum_{p \in \mathcal{P}^k} \varphi_p = \lambda^{nc}, \quad \forall k \in K, & (5.2) \\ f_a \geq \sum_{\substack{p \in \mathcal{P}^k: \\ a \in p}} \varphi_p, \quad \forall a \in A, k \in K, & (5.3) \\ f_a \leq C_a, \quad \forall a \in A, & (5.4) \\ \varphi_p, f_a \geq 0, \quad \forall a \in A, p \in \mathcal{P}^k. & (5.5) \end{cases}$$

We denote by Ω_{base} the set of feasible (throughput, flow) vectors defined by the constraints of NC_{base} (5.2)−(5.5) and by Ω°_{base}, the same set where the capacity constraints (5.4) are removed (relaxed). We will use these notations later to define our extended models.

5.3.3 A Single Multicast Tree Model

Spanning trees and Steiner trees have been studied for a very long time, but most of the times with the objective to minimize the cost of the tree. The usual model for multicast network throughput evaluation is the *Fractional Steiner Tree Packing* (FSTP) problem which is known to be NP-hard [47]. Note that, in this chapter, we use the term *tree*, although we should use the term *arborescence* since we use a directed graph. The classical models for FSTP are based on tree variables and can be solved using column-generation techniques but this model is not easy to modify to limit the number of trees.

Here, we provide a model for maximizing the throughput within a single multicast tree. Strictly speaking, it is hence not really a "classical" model, but it is largely inspired by existing models. Indeed, we use a directed Steiner tree model and adapt it to our objective of maximizing the flow from the common source towards all terminal nodes. This model is very similar to NC_{base} except that a set of binary variables are introduced to model the tree:

$$MC_{1tree} \begin{cases} \max \lambda^{tree}, & & (5.6) \\ (\lambda^{tree}, f) \in \Omega_{base}, & & \\ y(\delta^+(s)) \geq 1, & & (5.7) \\ y(\delta^-(v)) \leq 1, & \forall v \in S, & (5.8) \\ y(\delta^-(v)) = 1, & \forall v \in K, & (5.9) \\ f_a \leq C_a \times y_a, & \forall a \in A, & (5.10) \\ f_a \geq 0, y_a \in \{0,1\}, & \forall a \in A. & (5.11) \end{cases}$$

Constraints (5.7), (5.8) and (5.9) define the arcs used in the tree (at least one out-going arc from the source, at most one in-coming arc in each non-terminal node and exactly one in-coming arc in each destination node). Constraints (5.10) link the binary tree variables with the flow variables within a capacity constraint. We denote by Ω_{tree} the set of feasible (throughput, flow) vectors defined by this problem and by Ω°_{tree}, the same set where the capacity constraints are removed. Note that, with this model, we might endup with some extra arcs identified by y_a variables, but without any impact on the optimal solution (these arcs can easily be trimmed within a post-optimization phase).

5.4 Controlling the Transit across the Nodes

Since we are interested in tracking the functionalities of a node (does it code ? or replicate ?), we need to define a way to map these functionalities with the features of the nodes in our flow model. Figure 5.1 illustrates what we can directly derive from the NC_{base} model on a small example: optimizing the model, one obtains a maximum throughput value of 3 and the resulting aggregated flows f_a (as depicted in the Figure). We could hence assume that there is a solution with a throughput value of 3 and where 2 nodes are coding (nodes v_4 and v_5) and 5 nodes are performing multicast (nodes v_1, v_2, v_3, v_6 and v_7).

In this small example, it is easy to derive such a detailed solution (it is depicted

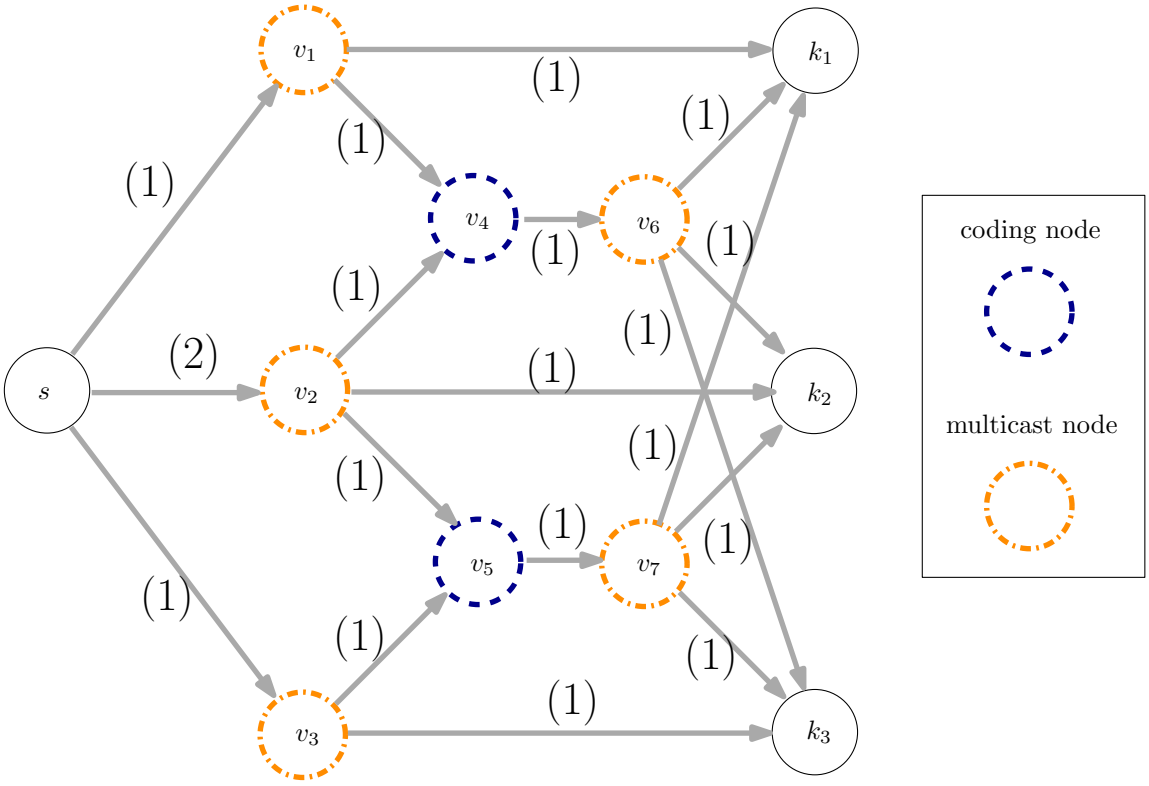


Figure 5.1: Results of NC_{base} model on a double butterfly instance: the values indicated in brackets are the f_a aggregated flow values (they all correspond to the arc capacities, in other words, all arcs are saturated).

in Figure 5.2). It is interesting to observe that node v_2 is replicating stream b on two outgoing links and forwarding stream c on a third link. Of course, node v_2 could also have coded the two streams and multicast the resulting combination on the three outgoing links. There are many different ways to achieve the maximum throughput value of 3.

Based on these observations, we characterize the nodes according to the flow balance between the incoming and outgoing links:

$$\Delta(f, v) = f(\delta^-(v)) - f(\delta^+(v)). \tag{5.12}$$

Given a flow f (resulting from a throughput λ), a node v will be called a "coding node" if $\Delta(f, v) > 0$, a "multicast node" if $\Delta(f, v) < 0$, and a "unicast" or "neutral node" if the flow is balanced.

In order to track or to force specific behavior on some nodes, we need to introduce additional binary variables: the variables $x_v^{nc} \in \{0, 1\}$ are used to identify the

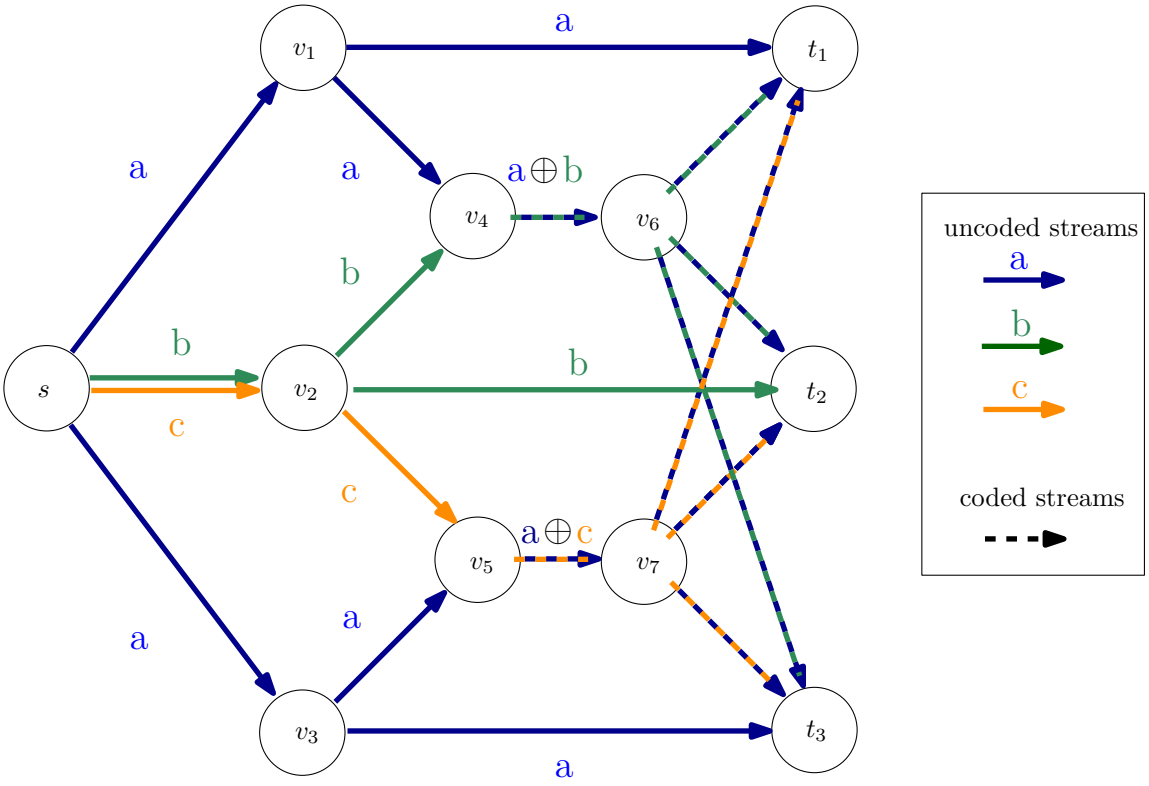


Figure 5.2: A translation of the aggregated flow results of Figure 5.1 using three unit streams a, b and c.

coding nodes and the variables $x_v^{mc} \in \{0, 1\}$ are used to identify the multicast nodes. Without loss of generality, we will assume here that the same node cannot perform both functions. This is also in line with our aim to minimize extra network management tasks. The general case can be mapped into our simplified case by a simple graph transformation, where each node is split into one input node and one output node (see Figure 5.3).

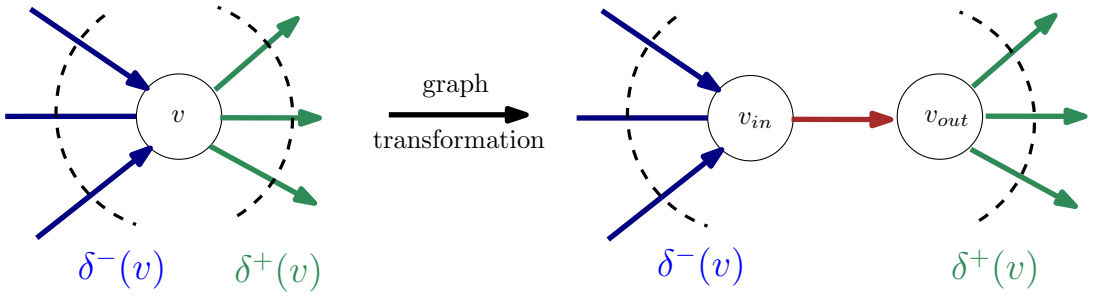


Figure 5.3: Transformation of a general graph so that each node can be tagged exclusively as "coding", "multicast" or "neutral".

5.5 Extended Throughput Maximization Models

5.5.1 Controlling the Number of Coding/Multicast Nodes

Based on the observation of the previous section, we are now ready to define our first extended model:

$$NC_{ext1}\begin{cases} \max \lambda, & & (5.13)\\ (\lambda, f) \in \Omega_{base}, & & \\ f(\delta^-(v)) - f(\delta^+(v)) \leq M x_v^{nc}, & \forall v \in S, & (5.14)\\ f(\delta^+(v)) - f(\delta^-(v)) \leq M x_v^{mc}, & \forall v \in S, & (5.15)\\ x_v^{nc}(S) \leq N^{nc}, & & (5.16)\\ x_v^{mc}(S) \leq N^{mc}, & & (5.17)\\ f_a \geq 0, & \forall a \in A, & (5.18)\\ x_v^{nc}, x_v^{mc} \in \{0,1\}, & \forall v \in S. & (5.19) \end{cases}$$

Constraints (5.14) and (5.15) allow to track or to switch on or off the "coding" and "multicast" property of the nodes. Constant M stands for a "big-M" (a value large enough to deactivate the constraint when the binary variable is set to 1). Since the capacities on the links are limited, we can easily derive upper-bounds for these excess or shortage values: we can indeed use the values $M = C(\delta^-(v))$ for (5.14) and $M = C(\delta^+(v))$ for (5.15). Note that the constraints $x_v^{nc} + x_v^{mc} \leq 1$, for each node $v \in V$, are implicitly satisfied by our model (because the left-hand-side of (5.14) and (5.15) cannot be strictly positive and strictly negative at the same time). In this model, our aim is to limit the number of coding and multicast nodes below given threshold values N^{nc} and N^{mc}. This limitation is obtained through the constraints (5.16) and (5.17).

Note that we could also consider a more general "budget" constraint, where given an available budget B, unit costs c^{nc} and c^{mc} for deploying network coding or multicast in a node, the total cost of deploying coding/multicasting should remain within the budget:

$$\sum_{v \in S} c^{nc} x_v^{nc} + \sum_{v \in S} c^{mc} x_v^{mc} \leq B. \tag{5.20}$$

In this model, we mainly consider the cost brought to network operators, that is why the decoding cost is not in the model. The next model in Section 5.5.2 will

consider the decoding cost on the end users' side.

It should be noted that this problem involves binary variables and is hence probably not as "easy" as NC_{base} model which is pure (continuous) linear programming problem. In fact, it has been proved in [55] that the related problem of minimizing the number of coding nodes given a throughput value is NP-hard. As a result, this reverse problem is also NP-hard (since the decision problems are the same).

In this first extended model, we assume that the multicast nodes can perform a partial replication of the incoming flows towards some (but not necessarily all) outgoing arcs. If we are more interested in modeling a broadcast-type behaviour, then all incoming traffic should be replicated over all outgoing links. To model broadcast nodes, we need to modify slightly our extended model. We use binary variable x_v^{bc} (instead of x_v^{mc}) together with constraints similar to (5.15), to allow and enforce broadcasting on certain nodes. With our flow model, we must ensure that, if broadcasting is activated on a node $v \in S$ ($x_v^{bc} = 1$), then all incoming flows are replicated towards ALL outgoing arcs. This can be achieved with the following constraint:

$$f_a \geq f(\delta^-(v)) - C(\delta^-(v))(1 - x_v^{bc}), \forall v \in S, a \in \delta^+(v). \tag{5.21}$$

When $x_v^{bc} = 1$, the constraint is active (by removing the last term), otherwise, it is inactive (because the right-hand-side becomes less than 0). Model NC_{ext2} is defined by replacing the variable x_v^{mc} by x_v^{bc} in NC_{ext1} model and by adding constraints (5.21).

5.5.2 Combining Pure Network coding With One or Two Multicast Trees

We now define new models to evaluate the throughput achieved by sending uncoded traffic on one or two multicast trees and simultaneously sending coded traffic throughout the network. We hence need to combine problem NC_{base} with problem MC_{1tree} and sum the two types of traffic in order to satisfy a global capacity constraint. The problem we define involves a parameter $\alpha \in [0, 1]$ indicating how the traffic should be split in the network coding subgraph and the

multicast tree:

$$NC1T(\alpha) \begin{cases} \max \lambda, & & (5.22) \\ \lambda = (1-\alpha)\lambda^{nc} + \alpha\lambda^{tree}, & & (5.23) \\ (\lambda^{nc}, f^{nc}) \in \Omega^{\circ}_{base}, & & (5.24) \\ (\lambda^{tree}, f^{tree}) \in \Omega^{\circ}_{tree}, & & (5.25) \\ f^{nc}_a + f^{tree}_a \leq C_a, & \forall a \in A, & (5.26) \\ f^{nc}_a, f^{tree}_a \geq 0, & \forall a \in A. & (5.27) \end{cases}$$

In this problem, we hence maximize a global throughput value λ that is shared among network coding and one multicast tree according to the parameter α. Note that, when $\alpha = 0$, the problem reduces to NC_{base} model and when $\alpha = 1$ the problem reduces to MC_{tree} problem.

To analyze the case where two distinct multicast trees are used together with network coded flow, we define the problem $NC2T(\alpha)$ in a similar fashion, except that two different sets of tree variables and constraints must be added within the same problem. The throughput constraints (5.23) are replaced by:

$$\lambda = \alpha\lambda^{nc} + (1-\alpha)/2\lambda^{tree1} + (1-\alpha)/2\lambda^{tree2}, \tag{5.28}$$

and the capacity constraints (5.23) are replaced by

$$f^{nc}_a + f^{tree1}_a + f^{tree2}_a \leq C_a, \ \forall a \in A, \tag{5.29}$$

where the three types of flows are summed up on each arc.

5.6 Computational Experiments on Some Butterfly-like instances

The various models presented in the previous sections have been coded with the modeling language embedded in the commercial solver Xpress MP. The Xpress Optimizer (Version 21.01.00) has then been used to solve to optimality various instances of the considered problems.

We have first tested our models on three small toy instances built around the famous butterfly topology. The first instance, called *butterfly_1* is the standard butterfly network (see Figure 5.4).

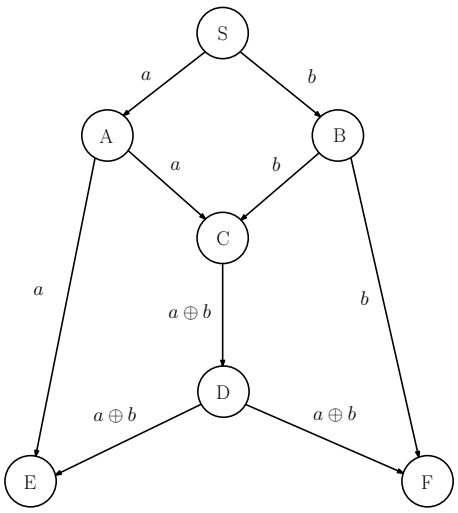


Figure 5.4: Butterfly Network

	N^{mc}						
N^{nc}	0	1	2	3	4	5	∞
0	1.333	2	2.667	2.667	2.667	2.667	2.667
1	1.333	2	2.667	2.667	3	3	3
	N^{bc}						
N^{nc}	0	1	2	3	4	5	∞
0	1.333	2	2.667	2.667	2.667	2.667	2.667
1	1.333	2	2.667	2.667	2.667	2.667	2.667
2	1.333	2	2.667	2.667	2.667	3	3

Table 5.1: Optimal throughput values obtained with models NC_{ext1} and NC_{ext2} on the *butterflyx2_1* instance.

The second, called *butterflyx2_1* is the double butterfly network depicted in Figure 5.1 (note that the capacity on arc (s, v_2) is 2 whereas it is 1 everywhere else). The third instance, called *butterflyx3_1* is a cyclic triple butterfly instance depicted in Figure 5.5 (all arc capacities are equal to 1).

Using our extended models NC_{ext1} and NC_{ext2}, we could evaluate the different throughput values obtained in *butterflyx2_1*. These values are reported in Table 5.1. We can see that the same value of maximum throughput (equal to 3) can be obtained using only one coding node and four multicast nodes. With less coding/multicasting nodes, the throughput decreases to 2.667 (8/3). Using only unicast, the throughput is equal to 1.333 (4/3).

If we observe again Figure 5.2, we see that the solution requires to deploy a rather complex multicast scheme (for instance on node v_2). It would be simpler, from

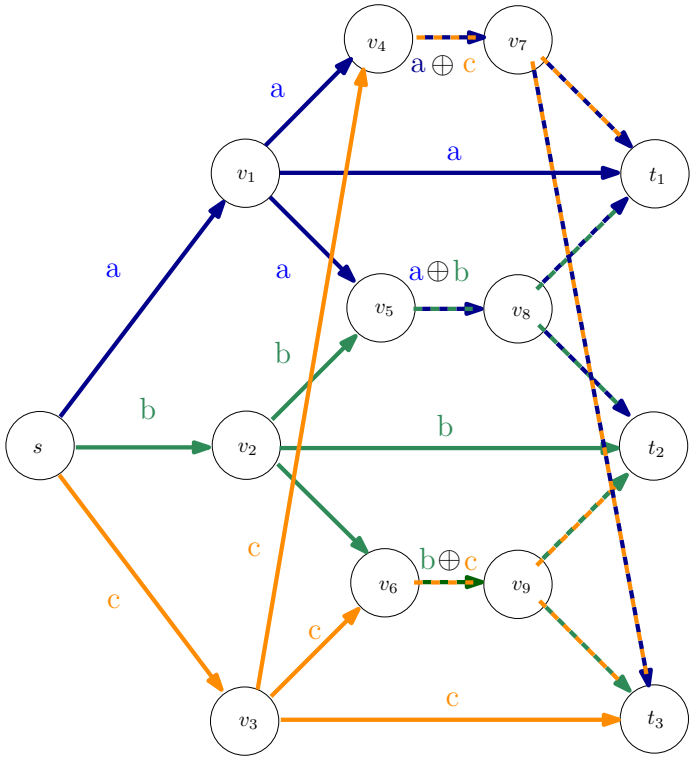


Figure 5.5: Cyclic triple butterfly instance.

an operational point of view, to use broadcast (limited to certain nodes) instead of multicast. Using model NC_{ext2}, we obtain slightly different results. We see for instance, that at least two network coding nodes are required to reach the optimal throughput.

Similar results for *butterflyx3_1* are reported in Table 5.2. We can observe that there is much more variety in the achieved throughput values. When using multicast replication together with network coding, the number of multicast nodes seems to have a much larger impact on the throughput than the number of network coding nodes. When using broadcast instead of multicast, we see that both functionalities need to be used together and the impact of the number of coding nodes becomes significant.

To evaluate the impact of a combined use of network coding together with a few multicast trees, we have used our models $NC1T(\alpha)$ and $NC2T(\alpha)$. The resulting throughput curves appear in Figures 5.6 (for one multicast tree) and 5.7 (for two multicast trees).

The network coding throughput values on the left of the x-axis ($\alpha = 0$) are the

	N^{mc}						
N^{nc}	0	1	2	3	4	5	∞
0	1	1.5	1.8	2	2	2	2
1	1	1.5	1.8	2.2	2.333	2.333	2.333
2	1	1.5	1.8	2.2	2.333	2.5	2.5
3	1	1.5	1.8	2.2	2.333	2.5	3

	N^{bc}						
N^{nc}	0	1	2	3	4	5	∞
0	1	1.333	1.5	2	2	2	2
1	1	1.5	1.667	2	2	2	2
2	1	1.5	1.8	2	2	2	2
3	1	1.5	1.8	2.2	2.333	2.5	3

Table 5.2: Optimal throughput values obtained with models NC_{ext1} and NC_{ext2} on the *butterflyx3_1* instance.

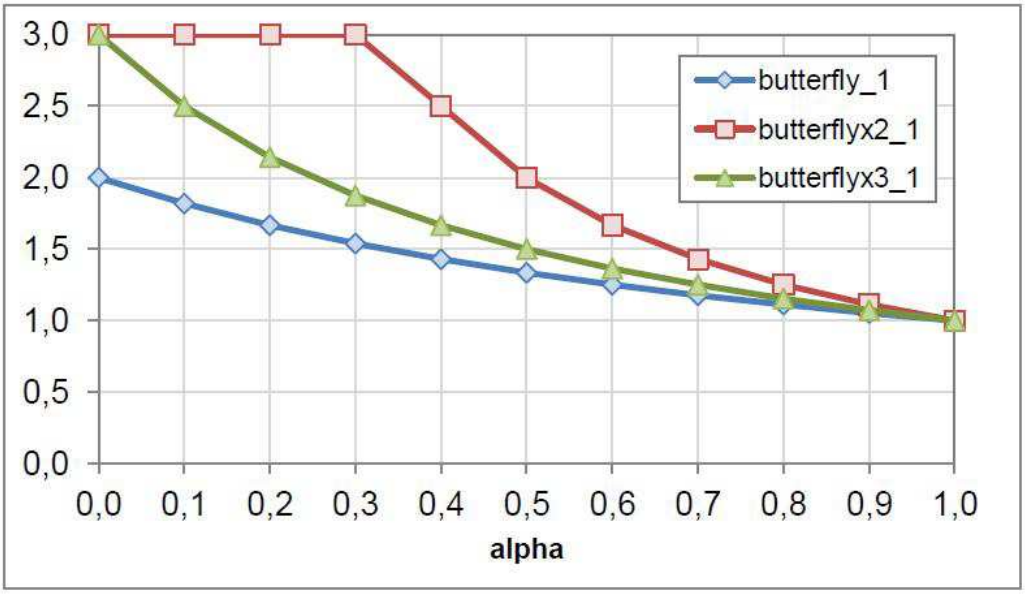


Figure 5.6: Throughput gain in butterfly network by mixing a single tree and network coding:Throughput $(1-\alpha)\lambda_{NC} + \alpha\lambda_{1T}$ obtained when splitting the flow over a network coding sub-graph and a single multicast tree.

same in both Figures but we see that, with two multicast trees, the multicast throughput on the right of the x-axis ($\alpha = 1$) increases from 1 to 2 for the *butterflyx2_1* instance. We also see, on the same instance, that with 30% multicast traffic (or less), network coding is able to manage the remaining traffic in order to reach the maximum throughput (of 3). The general trend of the curves is, as expected, decreasing (more traffic is sent on multicast trees, less throughput is achieved), but it also appears that the amount of network coding needed in order to have a significant impact on throughput is rather large (around 50%).

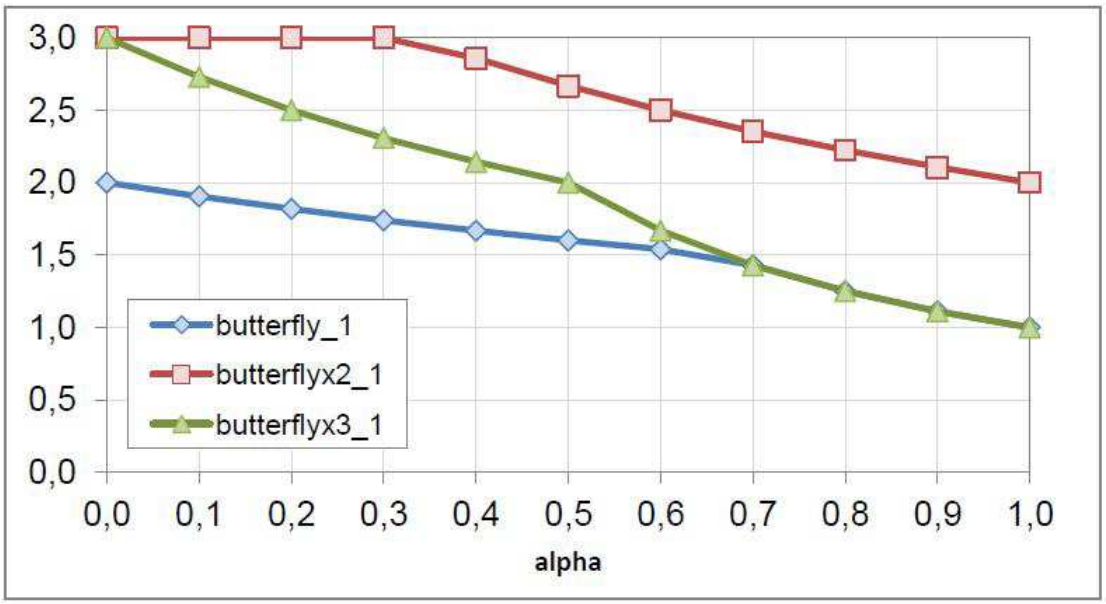


Figure 5.7: Throughput gain in butterfly network by mixing two multicast trees and network coding: Throughput $(1-\alpha)\lambda_{NC} + \alpha\lambda_{1T}$ obtained when splitting the flow over a network coding sub-graph and two multicast trees.

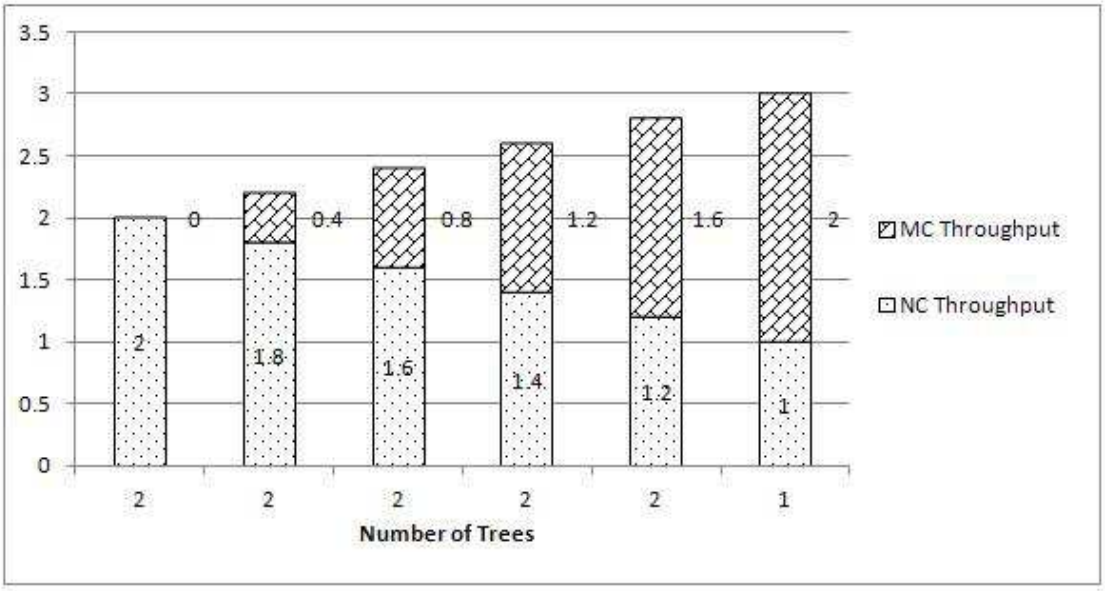


Figure 5.8: Various combinations of throughput values obtained on instance *butterflyx2_1* when progressively reducing the share of multicast traffic.

The final Figure 5.8 gives an interesting view of the benefit that can be expected from a limited introduction of network coding within a multicast network. Starting with two multicast trees, the throughput obtained is 2. By reducing the

throughput on these two multicast trees to a value of 1.8, the global throughput then goes to 2.2 by the introduction of network coding. The more the share of multicast throughput is reduced, the more the global throughput increases, thanks to network coding.

5.7 Computational Experiments on Randomly Generated Instances

Before giving numerical results, we first show an example of different optimal routings in a random generated graph in Figures 5.9, 5.10 and 5.11. These tree graphs are the screenshots of a random graph with 21 nodes, 120 links and 10 terminals but with different routing strategies. Figures 5.9, 5.10 and 5.11 apply respectively, pure network coding, single multicast tree, and strategic network coding in which single multicast tree and network coding are used.

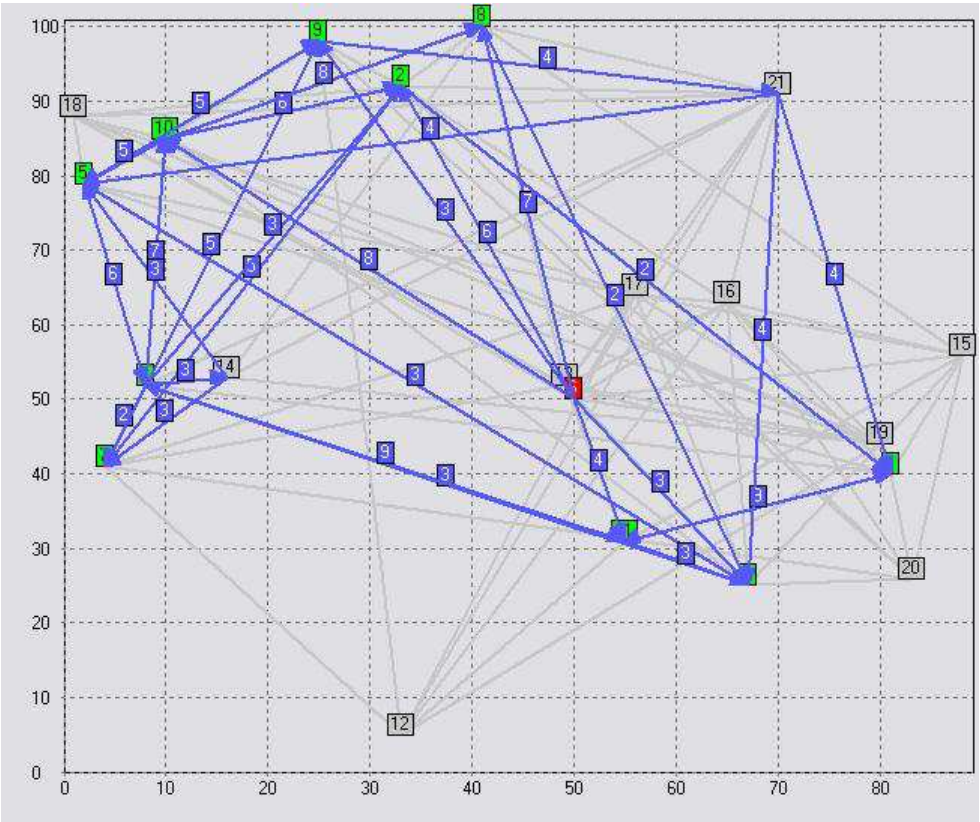


Figure 5.9: Optimal network coding flow in a random graph: example of graph with 21 nodes, 120 links and 10 terminals: a solution with network coding only ($\alpha = 0$, throughput $= 15$).

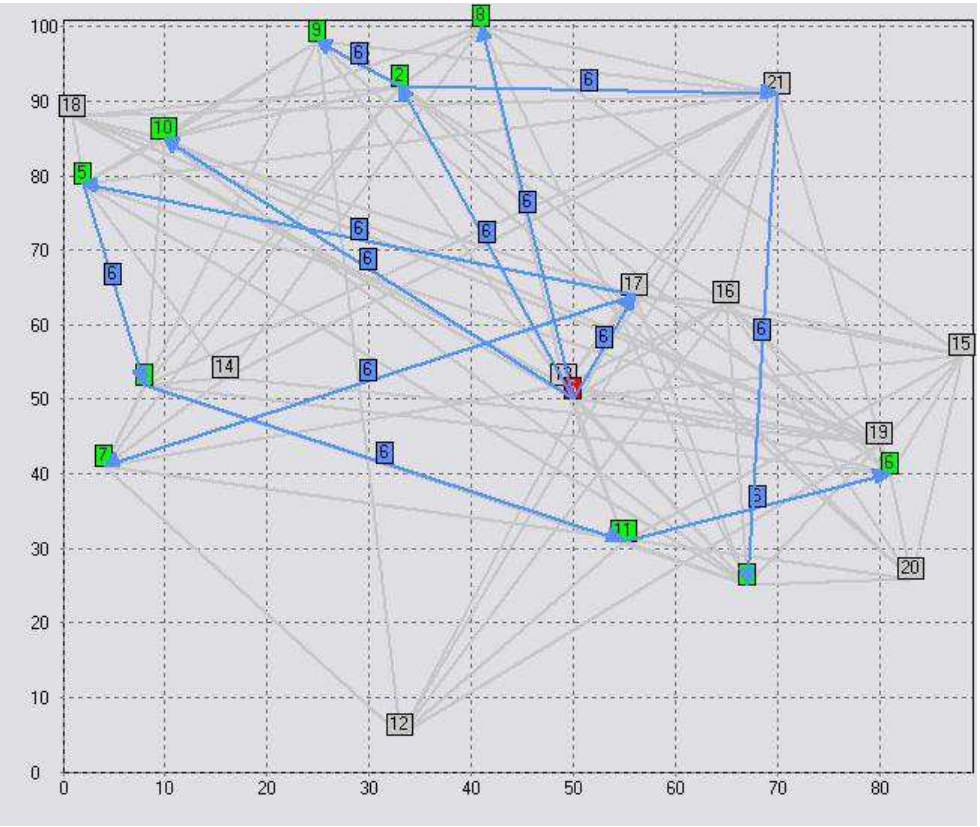


Figure 5.10: Optimal single multicast tree in a random graph: example of graph with 21 nodes, 120 links and 10 terminals: a solution with a single multicast tree only ($\alpha = 1$, throughput $= 6$).

From the graphs shown in Figures 5.9, 5.10 and 5.11, we make some observations: network coding still performs the best, in terms of throughput gain; the single multicast tree involves the least number of arcs in sending information, but the throughput declines dramatically compared to network coding; strategic network coding achieves a throughput very close to the one obtained by network coding, but the amount of network coding flow has a significant decrease. That is, indeed, the benefit that we expect from the mixed routing strategy, could maintain a relatively high throughput but also reduces complexity by using network coding.

For the numerical experiments, the instances have been randomly generated (in a similar fashion as the graph generator in Chapter 4) first by building a strongly connected graph (with one path between the source node and each one of the terminals) and then progressively increasing the density of the graph up to a required level. Arc capacities are randomly generated in the interval $[0, 10]$. For given sets (n, m, N_t), series of 100 instances have been randomly generated and average results are now reported.

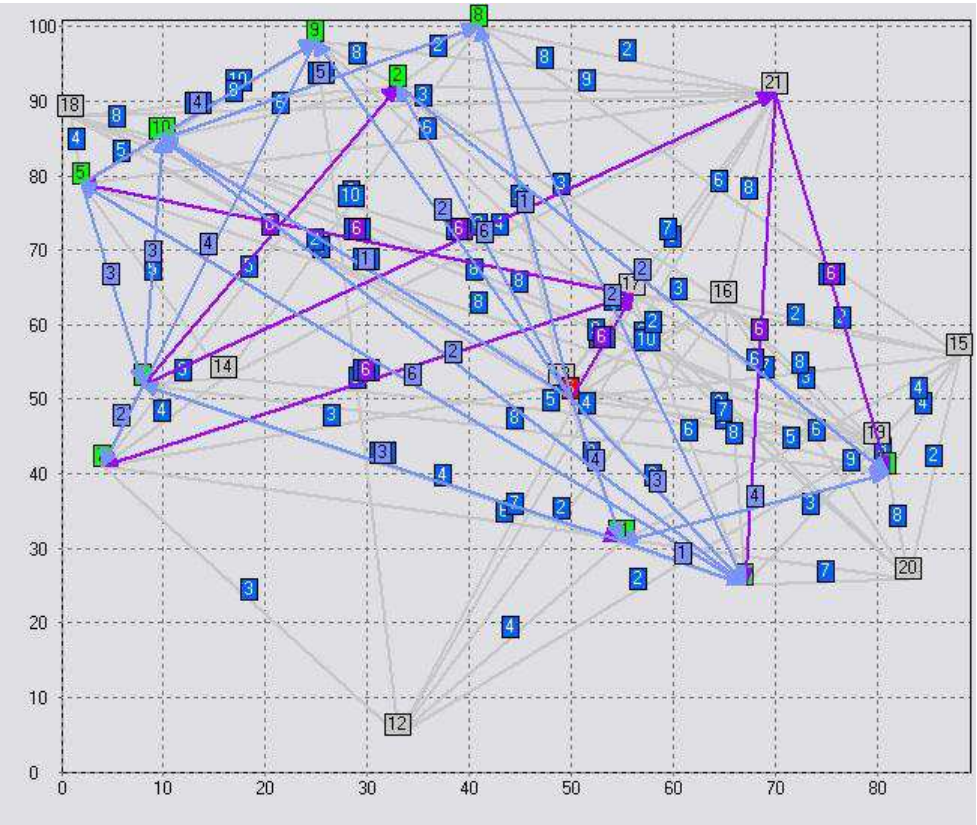


Figure 5.11: Optimal strategic network coding in a random graph: example of graph with 21 nodes, 120 links and 10 terminals: an intermediate solution ($\alpha = 0.5$), throughput = 12).

5.7.1 Bridging the Gap Between Network Coding and Multicast

Several experiments in Chapter 4 have shown that a multicast strategy can offer almost as much throughput as network coding, but the number of multicast trees needed can be large. Since this is not affordable in most practical settings, we consider here the case where only one multicast tree is used. In this case, there is a huge gap between the multicast and the network coding throughput. This is confirmed by our first series of computational experiments. Figure 5.12 shows the (average) relative throughput values obtained using also some intermediate strategies (100% represents the pure network coding throughput - not depicted in the figure):

- NC+BC: network coding with broadcast (on some nodes);
- NC(p=1)+BC: the same as NC+BC but with only one coding node;
- NC(p=2)+BC: the same as NC+BC but with only two coding node;

- MC(1t): multicast but with only one tree.

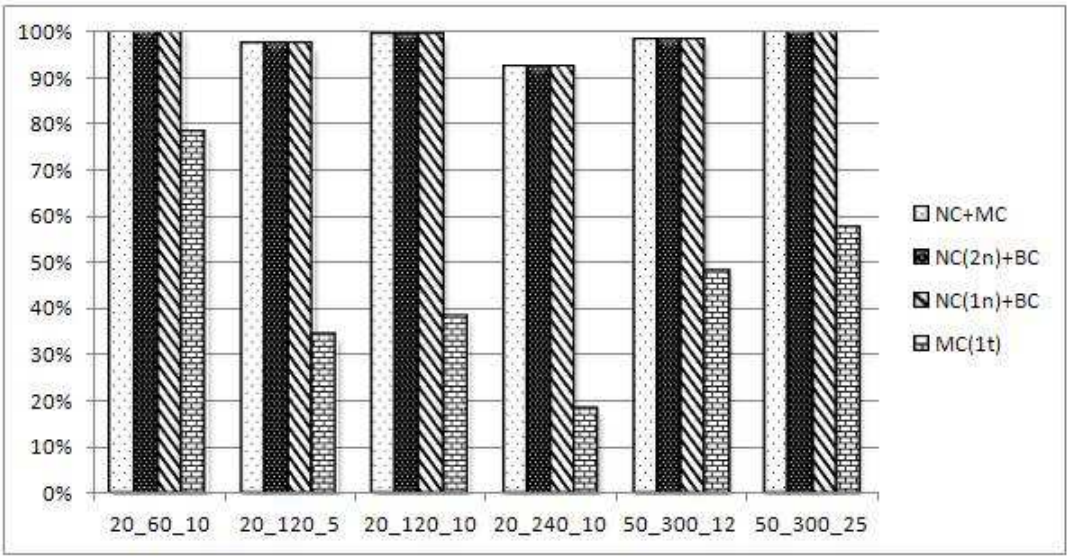


Figure 5.12: Percentage of the maximum throughput for different routing strategies: percentage of the maximum throughput observed when considering various coding/replication strategies (100% for NC), for various sets of instances.

We can make the following observations: throughput obtained with a single multicast tree MC(1t) is always much below the maximum (network coding) throughput. Imposing broadcast (BC) instead of partial replication (multicast) within a network coding scheme (NC) has a very small impact on the throughput. In the densest instances (20_240_10), the reduction in throughput can reach almost 10%, but such instances are not very realistic, since telecommunication networks are usually much sparser. Finally, and this is perhaps the most surprising observation, in almost all cases, a single coding node is sufficient to reach the best possible throughput value. This militates very much in favor of network coding because the cost of deploying coding over a single node might be low.

One step further along this line, we have compared the impact of the number of coding nodes with another important parameter for deploying network coding, namely the total capacity of the incoming links connected to each terminal node, which are often bottleneck links in telecommunication networks. In our model, we have chosen to consider instead the in-degree of each terminal nodes, which we have intentionally bounded to small values (1, 2 or 3 in our experiments).

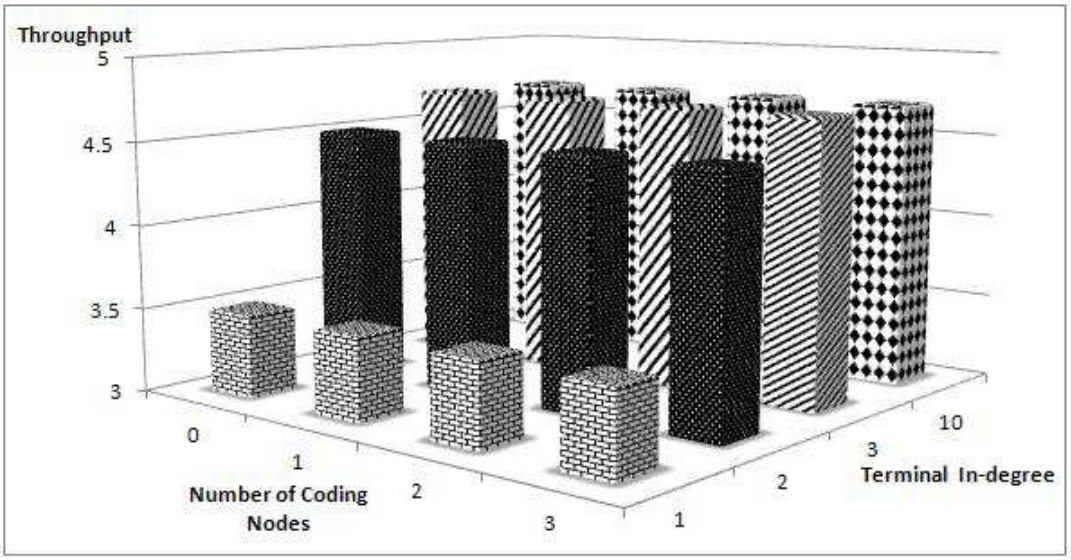


Figure 5.13: Average maximum throughput in a random graph when the coding nodes are limited: average maximum throughput observed on 20_60_10 instances when limiting the number of coding nodes and the in-degree of each terminal node.

It is obvious from Figure 5.13 that the number of coding nodes has (again) a very limited impact, whereas limiting the flows reaching the terminals has a more significant impact, especially when going from 1 to 2, because in our models, we assume no intra-flow coding (several streams coded within a same path), so the coding can only occur when at least two flows on two distinct interfaces can reach the terminal nodes. A finer analysis could be performed by taking into account intra-flow coding within our models.

5.7.2 Combining Network Coding with a Few Multicast Trees

In this set of computational experiments, we solve problem $NC1T(\alpha)$, for selected values of α, on several sets of instances.

The curves displayed in Figure 5.14 show the (average) percentage of the maximum throughput obtained (100% for NC). We see that, in most instances except the densest ones (20_240_10), it is already beneficial to introduce a small percentage of network coding traffic. For instance, there is a relative throughput improvement of 4% when switching 10% multicast traffic to network coding and this increase is more or less linear until reaching the maximum throughput (ob-

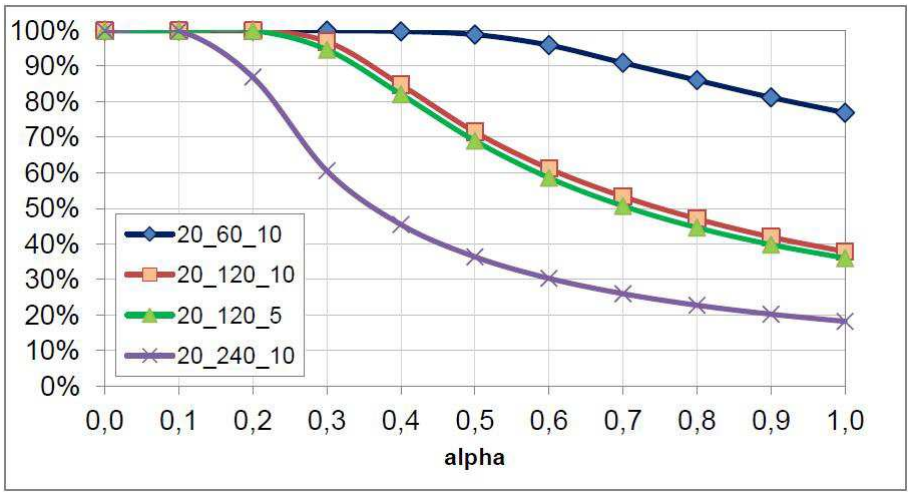


Figure 5.14: Throughput gain in random graphs by mixing a single multicast tree and network coding: percentage of the optimal (network coding) throughput obtained when splitting the flow over a network coding sub-graph and a single multicast tree.

tained for 50% or even 70% of network coding traffic). As a consequence, if the ratio of the cost of introducing network coding over the benefit obtained from increasing throughput is less than 1 over 2, it seems interesting to deploy some network coding within the network.

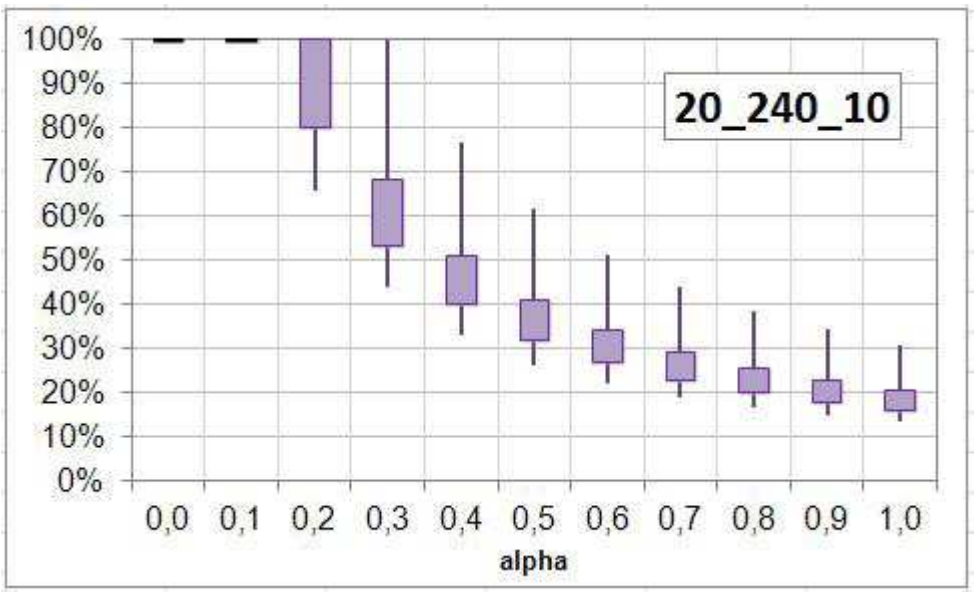


Figure 5.15: Statistical results on the percentage of throughput compared to network coding in a random graph: detailed solutions for instance 20_240_10 (boxes contain 50% of the instances, the extreme values [min, max] are represented by the segments).

Figure 5.15 gives some details, for the densest instance, about the spreading of different values around the average. If the range of extreme values is rather large, we can observe that 50% of the instances follow quite closely the trend announced by the average value, namely, that the impact of introducing network coding grows dramatically after one third of the traffic is already switched to network coding.

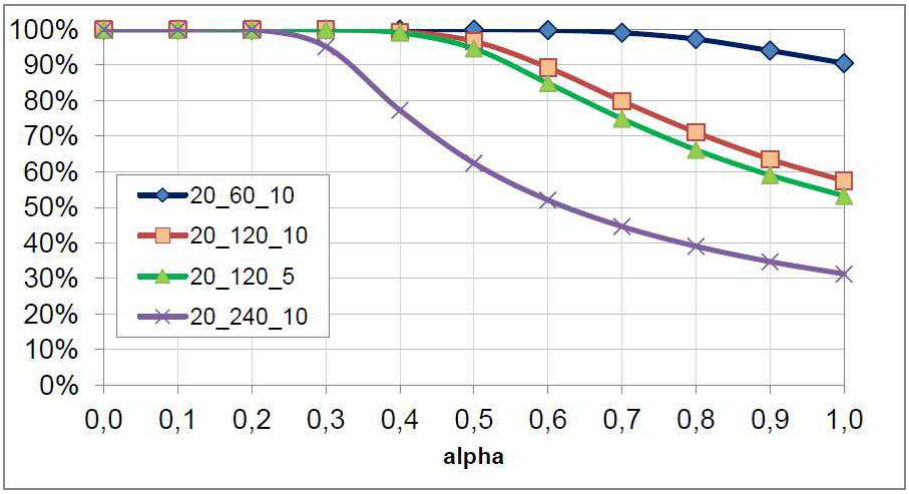


Figure 5.16: Throughput gain in random graphs by mixing two multicast trees and network coding: percentage of the optimal (network coding) throughput obtained when splitting the flow over a network coding sub-graph and two single multicast trees.

If we consider the case where two multicast trees can be used in parallel together with network coding, we need to solve problem $NC2T(\alpha)$. Note that, in our model, the share $\alpha\lambda$ of multicast traffic is equally split on the two trees ($\alpha\lambda/2$ on each tree). The results are depicted in Figure 5.16. We see that the difference between pure network coding throughput ($\alpha = 0$) and pure multicast throughput ($\alpha = 1$) is somewhat reduced, but the overall tendencies and observations remain the same, i.e., introducing a small amount of network coding is already beneficial, but the benefit should be compared to the cost incurred. To obtain most of the network coding benefit, more than 50% of the traffic should be handled with network coding. Observe that in the sparsest instances (20_60_10), the optimal throughput is already achieved with as little as about 30% of coded traffic.

5.7.3 Improving Throughput on Multicast Trees

To get more insight on these results, we have also performed some tests to evaluate the minimal level of network coding required to reach the maximum throughput.

These experiments are performed on random instances with unit capacities.

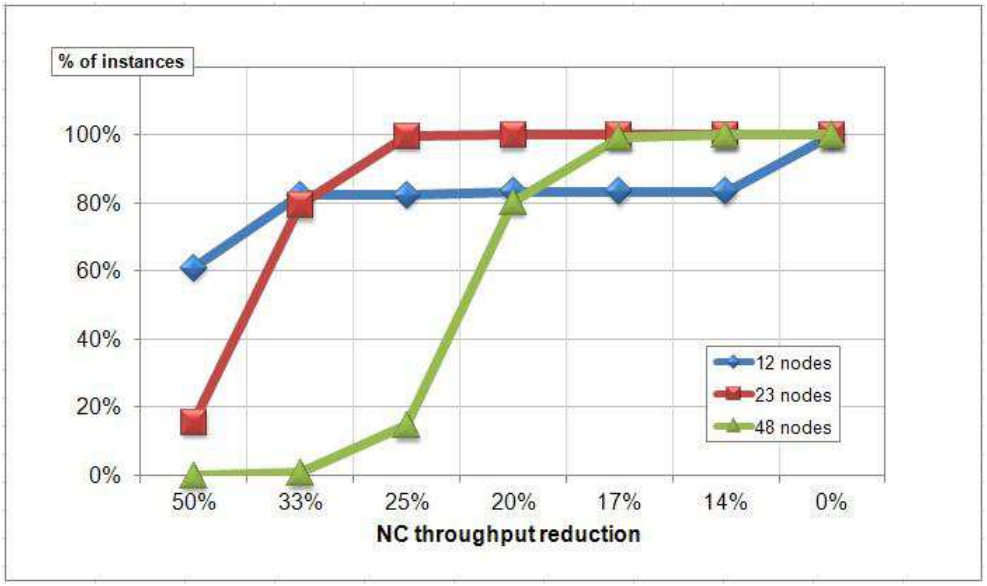


Figure 5.17: Cumulative number of instances (% over 300 instances) as a function of the network coding reduction to keep the same throughput and while using ONE multicast tree.

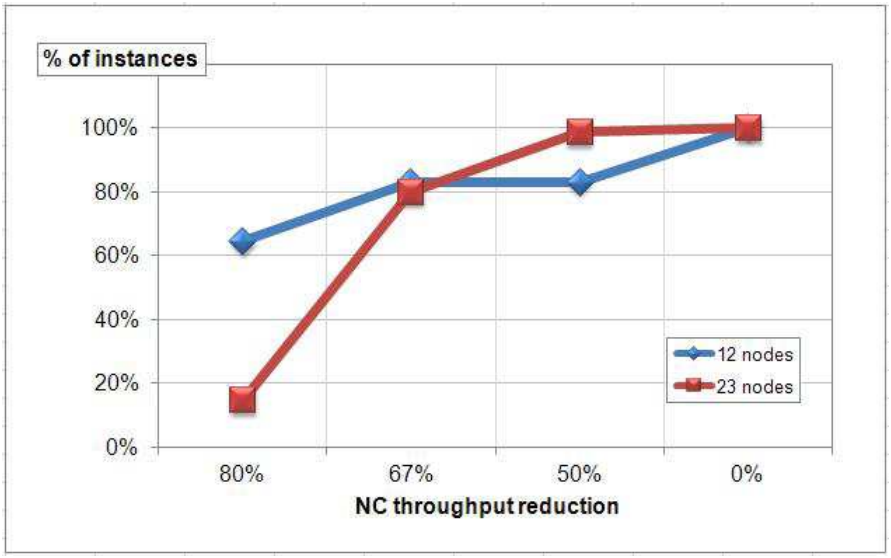


Figure 5.18: Cumulative number of instances (% over 300 instances) as a function of the network coding reduction to keep the same throughput and while using TWO multicast trees.

Figure 5.17 shows the repartition of instances where different reductions can be applied on the network coding share when introducing a single multicast tree so as to keep the same maximum throughput. We see that, for the smallest instances, in more than 60% of the cases, the share of network coding can be reduced by

50%, while in some cases, the reduction is smaller (sometimes, the use of a multicast tree does not even leave any space for coded traffic). When the number of nodes increases, the tendency is somehow shifted to the right, meaning that, on the average, less gain can be expected from the network coding contribution. For instances with 48 nodes, the network coding contribution can always be reduced by more than 17% and about 20% on the average. This is more or less in line with the previous observations on graphs with one unit capacities.

The same type of analysis was performed when two multicast trees are used (only on the 12 node and 23 node instances). The results are depicted in Figure 5.18. We see that in this case, the network coding contribution can be reduced by a much larger extent, with an average of about 67% for the largest considered instances.

5.8 A Word about the Efficiency of Our Models

Model NC_{ext1} is rather general and includes several well-known special cases: when $N^{nc} = N^{mc} = 0$ (where N^{nc} and N^{mc} denotes the number of coding nodes and multicast node, respectively), NC_{ext1} reduces to a standard multicommodity-flow model known as *Maximum Concurrent Flow* problem. When $N^{nc} = 0$ and $N^{mc} = n$ (each node is a multicast node) then NC_{ext1} should model the maximum throughput problem in a pure multicast network. This problem reduces to the *Fractional Steiner Tree (in our case, arborescence) Packing* (FSTP) problem which is known to be NP-hard [47]. Using our model to evaluate the multicast throughput, we end up with a pure linear formulation (with a polynomial number of variables and constraints):

$$MC \begin{cases} \max \lambda, & & (5.30) \\ \sum_{p \in \mathcal{P}^k} \varphi_p \geq \lambda, & \forall k \in K, & (5.31) \\ \sum_{\substack{p \in \mathcal{P}^k: \\ p \ni a}} \varphi_p = f_a \leq C_a, & \forall a \in A, k \in K, & (5.32) \\ f(\delta^-(v)) \leq f(\delta^+(v)), & \forall v \in S, & (5.33) \\ \varphi_p, f_a \geq 0, & \forall a \in A, p \in \mathcal{P}^k. & (5.34) \end{cases}$$

This problem can hence be solved in polynomial time, and thus, unless P=NP, it is not a valid model for FSTP. However, any solution of FSTP yields a valid solution for MC. It follows that MC is a relaxation of FSTP and hence provides

an approximate solution to FSTP (upper-bound).

Instances				MC		FSTP			
n	m	$\|S\|$	$\|K\|$	λ	N_{MC}	λ	N_{tree}	N_{MC}	N_{diff}
20	50	1	10	3.43	1.35	3.43	1.27	1.10	0
50	150	1	10	2.74	3.04	2.74	1.09	6.95	0
50	150	1	20	2.59	3.38	2.59	1.10	5.64	0
10	40	3	3	4.94	1.65	4.94	5.94	1.79	0
20	60	3	5	1.72	2.24	1.72	3.84	3.98	0

Table 5.3: Comparison of MC and FSTP solutions on series of 1000 randomly generated instances.

We have tried to evaluate empirically the quality of this approximation by several numerical experiments over series of randomly generated instances. We have used a tree generation algorithm to solve exactly FSTP which has already been introduced in Chapter 4, Section 4.4.3 (the pricing sub-problem is a Steiner tree problem that we solve using a standard Mixed Integer Programming formulation). The results are provided in Table 5.3. Each line describes average results over 1000 randomly generated instances of the types described by the first four columns (number of nodes, number of arcs, number of sources, number of terminals). For each one of the two models, we report λ the average maximum throughput and N_{MC} the average observed number of multicast nodes. For FSTP we also report N_{tree}, the average number of trees used in the solutions. Finally, N_{diff} represents the number of instances (over 1000) where the two solutions differ.

The first striking observation is that the solutions are always the same for the two models. MC seems hence a very good way to compute, in a heuristic fashion, the optimal value of FSTP. The quality of MC model should be investigated from a theoretical point of view, but since the purposes in this chapter is to evaluate throughput with limited number of coding and multicast nodes, we will here rely on this approximate model and its generalization NC_{ext1}. Note that the model NC_{ext2} is however a valid (and exact) model for the case when broadcast is considered instead of multicast.

5.9 Conclusion

In this chapter, we have carefully measured the impact of using jointly the network coding paradigm together with multicast routing, but keeping the setting

realistic from a network operation point of view, with a limited number of multicast trees and a limited number of coding nodes. We have proposed several mixed-integer programming models allowing to analyze various aspects of a careful and restricted introduction of network coding functionalities within an existing multicast network.

Our main findings are that a small introduction of network coding is already beneficial from a throughput point of view. However, to obtain the full benefit of network coding, a significant part (about 30% for the sparsest graphs and up to 50% for denser graphs) of the multicast traffic must be processed by network coding. The main cost incurred while deploying network coding lies on the decoding part near or at the terminal nodes. The cost of coding is negligible since we show (but this had already been perceived by other authors [53, 55]) that a very small number of coding nodes are needed (in our experimental setting, most of the time, a single coding node is enough to obtain the full coding benefit).

A future line of research would be to investigate more closely the economical aspects of a practical network coding deployment by taking into account real costs of equipments.

Chapter 6

Constrained Transportation Problem for Distributed Storage System

There are two ways to live your life, one is as though nothing is a miracle, the other is as though everything is a miracle.

-Albert Einstein (1879-1955)

6.1 Summary

After the introduction of network coding, the network coded distributed storage system receives a lot of attentions. In this chapter, we study a routing problem in such system which stores precoded content. Without other assumptions, the problem can be modeled as a transportation problem. It is a well-known problem where available supplies must be shipped to demand point through a bipartite graph. The problem of minimizing the sum of transportation cost is the most classical one, but some variants with different objectives have also been considered. In this chapter, we consider the problems with these objectives as well as the additional degree constraints introduced to the nodes as there is limited access to single servers in the system. Such constraints (or related ones) have already been considered in more general flow problems [22, 74], but combined with a transportation problem, the resulting problem becomes a combination of two

combinatorial problems, which are transportation problem and matching problem. To the best of our knowledge, this combined problem seems to have been very little investigated in the literature. We start here a new analysis of these problems, then we deliver some preliminary results and propose a Lagrangian decomposition approach.

6.2 Introduction

The Transportation Problem with cost minimization objective is a classical problem: in a bipartite graph, supply nodes must be connected to client nodes so that the flow or transshipment cost is minimized. But other objectives have also been considered: in the Minimax version, the maximum over all arcs of the transshipment cost is minimized; in the bottleneck version, it is the maximum cost over all used arcs that is minimized. Such objective are often used to model transshipment *delays* for instance, before all raw material is received by a factory which has to process it. Note that, the term *Minimax* is different from the one *Minmax* which is introduced in previous chapters. Minimax is used to represent a cost problem, where the cost can be considered as transmission latency, but according to Chapter 3, Minmax is used to represent a throughput problem. Moreover, Minimax is also a state-of-the-art definition given in [7].

The application that motivates our interest for these problems stems from the telecommunication field. Indeed, such problems can be used to model content distribution system in which various contents are transmitted from a set of servers to a set of clients. In this setting, the delay to receive information is of vital importance because it is one of the main component of the Quality of Service. Another important feature in these problems, is the fact that each server (and to a less extent, each client) cannot maintain a large number of simultaneous connections, or, in other words, cannot transmit data to many clients at the same time. It follows that we have to introduce degree constraints in the transportation model. It appears that, to the best of our knowledge, such degree constrained transportation models have been very little studied up to now.

6.3 Problem Statement

We denote by $I = \{1, \ldots, n_I\}$ the set of servers and by $J = \{1, \ldots, n_J\}$ the set of clients. Each server $i \in I$ has an available supply of a_i in data content and each client $j \in J$ has a demand requirement of b_j. We assume each server i can serve each client j with a unit cost $c_{ij} > 0$ (which can represent a distance, a delay or an energy consumption). We assume that the original data has been coded and the resulting coded data chunks are arbitrarily stored in the various servers up to their a_i levels. As a result, even if the clients' requirements correspond to some of the original data (with a total volume summing up to b_j), these data can be recovered from any amount of b_j received data: indeed, standard Network Coding results ensure that the original data can be recovered, with a very high probability, by decoding the coded data. Readers who are interested in the details of the network coding applications and mechanisms applied in distributed storage system can be referred to Chapter 2, Section 2.5 and some references mentioned therein. In practice, we should assume a small overhead Δ is needed to ensure the desired result, but it does not change the optimization problems at all since this overhead is present in both the supplies and the demand. As a result, the setting we have just described corresponds to a standard transportation problem: if x_{ij} denotes the non-negative quantity sent by server i to client j, we then have the usual constraints:

$$\sum_{j \in J} x_{ij} = a_i, \quad \forall i \in I, \tag{6.1}$$

$$\sum_{i \in I} x_{ij} = b_j, \quad \forall j \in J, \tag{6.2}$$

$$x_{ij} \geq 0, \qquad \forall i \in I, j \in J. \tag{6.3}$$

Using standard graph conventions, the transportation problem can be modeled with a bipartite directed graph $G = (V, A)$ where the set of nodes is $V = I \cup J$ and A is the set of all arcs between I and J (or a subset of it if we want to forbid some connections). We denote by $n = n_I + n_J$ the total number of nodes and m the number of arcs (most of the time, we will assume that G is a complete bipartite graph, i.e., $m = n_I \times n_J$). In a directed graph G, $\delta^+_G(v)$ (resp. $\delta^-_G(v)$) denotes the set of arcs having node s as tail (resp. head). Subscript G will be omitted if the graph considered is clear from the context. We have to assume that the total amount of supplies is equal to the total amount of demands (otherwise,

we can add a dummy node to absorb excess of supply or demand):

$$\sum_{i \in I} a_i = \sum_{j \in J} b_j. \tag{6.4}$$

We denote by T this common quantity. If one wishes to minimizes the total transportation cost:

$$(obj_1) : \min \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}, \tag{6.5}$$

then this finishes to define the standard Transportation Problem (TP) for which there exists plenty of efficient algorithms (see for instance [80]).

Since, in this chapter, we are aiming at designing a content distribution system in a telecommunication setting, there are other objective functions to consider which are much more relevant in this context. For instance, if c_{ij} represents the delay for a connection between server i and client j, then objective obj_1 amounts to minimize the average delay. However, from the client point of view, it seems more relevant to measure and minimize the total delay needed before a client j has received all his required data. Indeed, since we assume the original information is coded within the data stored in the servers, it is very likely that a client will need all data (or most of it) before he can start to decode. Hence the relevant delay metric for a client j is the delay needed to receive his last data chunk. If the storage system manager wants to make all clients happy, he will consider minimizing the maximum of all delays:

$$(obj_2) : \min \max_{i \in I, j \in J} c_{ij} x_{ij}. \tag{6.6}$$

The transportation problem with objective obj_2 is known as the *Minimax Transportation Problem* (MTP). Using the specific structure of the transportation problem, several polynomial time algorithms have been proposed for MTP (see, for instance, [7]). In some networks (for instance, optical networks), the transmission delay can be essentially considered independent of the quantity of data transmitted. In this case, the delay perceived by a client j is the maximum of all delays among links i, j used to carry some data. The objective function then becomes:

$$(obj_3) : \min \max_{i \in I, j \in J} \{c_{ij} : \ x_{ij} > 0\}. \tag{6.7}$$

Transportation problems with objective obj_3 are known as *Bottleneck Transportation Problems* (BTP) [19, 33, 40].

Note that interesting variants of these minmax delay problem consists in minimizing the average delay perceived by the clients:

$$(obj_{2'}) : \min \sum_{j \in J} \max_{i \in I} c_{ij} x_{ij}, \tag{6.8}$$

$$(obj_{3'}) : \min \sum_{j \in J} \max_{i \in I} \{c_{ij} : \ x_{ij} > 0\}. \tag{6.9}$$

Finally, in order to stick to our practical setting, we need to introduce additional constraints into the transportation model. Indeed, data servers cannot maintain simultaneously alive a large number of connections. As a result, we wish to introduce constraints limiting the number of arcs used by each server. For this purpose, we first need to introduce binary variables $y_{ij} \in \{0, 1\}$ together with a set of constraints linking these new variables with the continuous x variables:

$$x_{ij} \leq \overbrace{\max\{a_i, b_j\}}^{M_{ij}} y_{ij}, \ \forall i \in I, j \in J, \tag{6.10}$$

stating that arc (i, j) is used ($y_{ij} = 1$) as soon as it carries some traffic ($x_{ij} > 0$). Then, we can use these binary variables to introduce in the model the so-called degree constraints to limit the number of clients each server can be connected to:

$$\sum_{j \in J} y_{ij} \leq d^+(i), \ \forall i \in I. \tag{6.11}$$

Using standard graph notation, we will sometimes write $y(\delta^+(i))$ for the left-hand-side of the above constraints.

Similarly, we might consider degree constraints for the client nodes:

$$\sum_{i \in I} y_{ij} \leq d^-(j), \ \forall j \in J. \tag{6.12}$$

We denote by d^+_{max} and d^-_{max} the maximum of each set of degree bound:

$$d^+_{max} = \max_{i \in I} d^+(i), \ d^-_{max} = \max_{j \in J} d^-(j). \tag{6.13}$$

We can now define the set of solutions for our degree-constrained transportation problems:

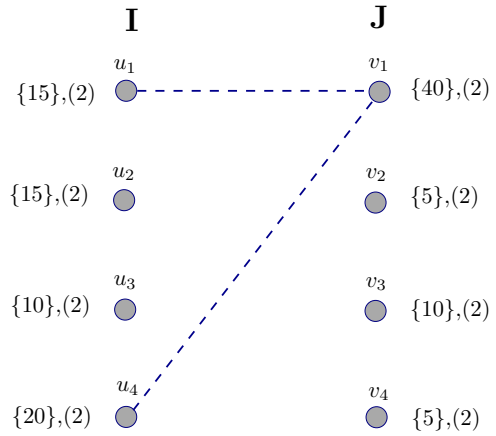
$$\mathcal{S}(a, b, d) = \left\{(x, y) \in \mathbb{R}^m_+ \times \{0, 1\}^m : \ (6.1) - (6.3), (6.10) - (6.12)\right\}. \tag{6.14}$$

Considering the solution set $\mathcal{S}(a, b, d)$ together with the various objective functions, we can define three new transportation problems, namely d-TP with obj_1, d-MTP with obj_2 and d-BTP with obj_3.

6.4 Relationship with b-matching and Feasibility Conditions

It is well-known that the standard transportation problem TP with supply a and demand b is nothing else than a fractional perfect (a, b)-matching problem. ("fractional" means that the variables are continuous and "perfect" means that the supply and demand constraints are satisfied with equality).

In our problems, the additional degree constraints restrict the set of arcs (used to carry some traffic) to be a feasible solution of simple d-matching problem ("simple" means that the arc variables should be in $\{0, 1\}$). As a consequence, we consider problems with two embedded matching problems, both being individually easy to solve, but the combination being generally much harder (this will be detailed in the next section).



	v_1	v_2	v_3	v_4	a_i	d_i^+
u_1	×				{15}	(2)
u_2					{15}	(2)
u_3					{10}	(2)
u_4	×				{20}	(2)
b_j	{40}	{5}	{10}	{5}	{60}	
d_j^-	(2)	(2)	(2)	(2)		

Figure 6.1: Infeasible instance of d-TP(1): the supply $\{a_i\}$ and demands $\{b_j\}$ are indicated within brackets and the degree bounds, (d_i^+) or (d_j^-) in parenthesis.

To give a first hint of the hardness of these problems, we can already observe that the combination of transportation and degree constraints can easily make the problem infeasible. Figure 6.1 shows an example of an infeasible problem. In this figure, there are two set of nodes: I and J where $u_i \in I, i = 1, \ldots, 4$

and $v_j \in J, j = 1, \ldots, 4$ denote supply nodes and demand nodes, respectively. The supply $\{a_i\}$ and demands $\{b_j\}$ are indicated within brackets and the degree bounds, (d_i^+) or (d_j^-) in parenthesis. The grid is an alternative way to represent the degree constrained transportation problem more explicitly. In each row, we assign some values in the cells but the sum of the values must be less than corresponding a_i in the given row. In addition, we can only fill values in two cells in one row, and leave other cells blank according to the degree constraint in this example. If one cell is filled with the total supply in that row, we use $\times$ instead of a specific number to represent the value we assign. Note that due to the degree constraint, at most two out four in each column can exist values or crosses. Although we fill the two largest supplies in the first column, the demand $d_1 = 40$ still cannot be satisfied.

This suggests to impose the condition that the supply of a subset of servers is not strictly larger than the maximum of possible demand spanned by the subset (and vice-versa):

Lemma 6.4.1. *Consider a feasible transportation problem d-TP. Then the two following (symmetrical) conditions hold:*

1. *For all* $S_1 \subseteq I$, $a(S_1) \leq \max\{b(S_2) : \ S_2 \subseteq J, |S_2| \leq d^+(S_1)\}$,
2. *For all* $S_2 \subseteq J$, $b(S_2) \leq \max\{a(S_1) : \ S_1 \subseteq I, |S_1| \leq d^-(S_2)\}$,

where we use a common notation $a(S_1) = \sum_{i \in S_1} a_i$ and $b(S_2) = \sum_{j \in S_2} b_j$.

Another immediate condition is that there exists a solution of the simple d-matching problem that cover all vertices. In the case of a complete bipartite graph (the case we consider here), this simply amounts to verify the condition:

$$d^+(I) \geq |J| \text{ and } d^-(J) \geq |I|. \tag{6.15}$$

However, this condition is not sufficient to guarantee the existence of a feasible solution in $\mathcal{S}(a, b, d)$. Consider, for instance, the example of Figure 6.2: it is easy to check that the condition of lemma 6.4.1 is satisfied. This example consists of three supply nodes and five demand nodes, and only two supply nodes u_1 and u_2 have degree bounds both of which are 2. We can observe that the supply node u_1 must deliver its supply to the clients node v_1 and v_2, because the demand of any two other combination of clients is less than $a_1 = 50$. But, in this case, the remaining demand of node v_2 is 10, and the supply node v_2 cannot find two

clients with a remaining demand of $a_2 = 40$.

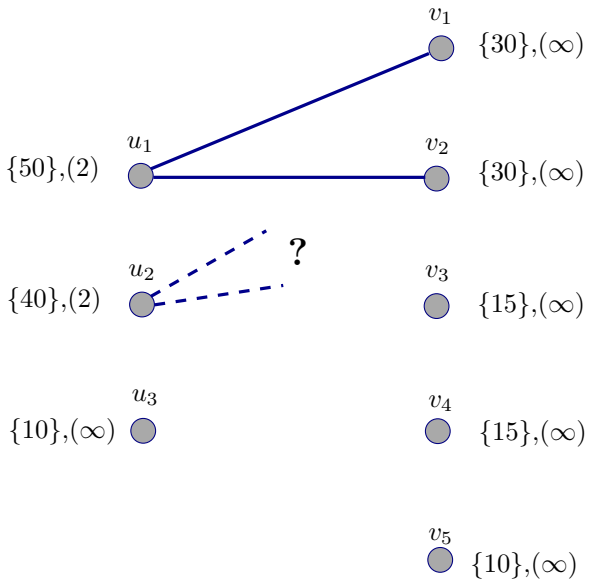


Figure 6.2: Infeasible instance of d-TP(2): the supply $\{a_i\}$ and demands $\{b_j\}$ are indicated within brackets and the degree bounds, (d_i^+) or (d_j^-) in parentheses.

It is hence reasonable to consider another optimization problem which aim is to check whether $\mathcal{S}(a, b, d)$ contains a feasible solution or not. For instance, we can consider the objective function consisting in maximizing the flow between I and J:

$$(obj_4) : \max \sum_{i \in I, j \in J} x_{ij}, \tag{6.16}$$

We denote by d-FTP this maximum Flow Transportation Problem defined over the set $\mathcal{S}(a, b, d)$ but with constraints (6.1) and (6.2) relaxed as inequality ($\geq$) constraints. If the optimal value of this problem is T, then all demands can be satisfied by all supplies on at least one solution, and our three previously defined problems are relevant.

Note that d-FTP can be related to a maximum multicommodity-flow, a problem defined as one possible generalization of the maximum-flow problem. One other popular generalization of the maximum-flow problem is the maximum concurrent flow problem, in which a common share of all commodities is maximized and shipped into the network. This approach can be applied to our degree-constrained problems. Consider for instance, the problem defined over $\mathcal{S}(a, b, d)$ with the objective function

$$(obj_5) : \max \lambda, \tag{6.17}$$

where $\lambda \geq 0$ is an additional variable, and where constraints (6.1) and (6.2) are changed into:

$$\sum_{j \in J} x_{ij} \geq \lambda a_i, \quad \forall i \in I, \tag{6.18}$$

$$\sum_{i \in I} x_{ij} \leq b_j, \quad \forall j \in J. \tag{6.19}$$

Another problem can be defined by inverting the role of the two constraint sets. We denote by d-CTP$_a$ and d-CTP$_b$ these two maximum concurrent transportation problems.

6.5 Complexity Issues

The classical Transportation Problem (TP) is one of the oldest OR problem and has hence been studied extensively (see, for instance, [73, 80]). Several methods in less than $O(n^3 \log n))$ have been proposed [75] and a linear-time algorithm is given in [45] for the case where the number of sources is fixed.

The Minimax Transportation Problem (MTP) has been proposed in [7] together with two algorithms, a parametric algorithm exploiting the specific structure of the transportation problem and a primal-dual maximum flot algorithm running in $O(n^4)$.

The Bottleneck Transportation Problem (BTP) was first considered in [40]. Several approaches have been proposed for BTP: a Hungarian method in [33], and an augmenting path method in [19].

When bounds on the degree come into play, the problems change from Linear Programming (LP) problems to mixed Integer Programming problems, which are normally much harder than LP. There are very few references on network flow problems where the number of arcs leaving a node and carrying some flow are restricted. In [23], the authors consider *d-furcated* network flow problems. A d-furcated flow is a flow that is forwarded on at most d outgoing arcs from every node. Hence, the case $d = 1$ corresponds to unsplittable flows, the case $d = 2$ corresponds bifurcated flows and when d is very large (larger than the maximum out-degree), the problem reduces to a standard fractional (unconstrained) flow. The problem considered in [23] is a single-sink multicommodity flow where the goal is to minimize the maximum flow (called congestion or load) over every

node, which is basically equivalent to a maximum concurrent flow problem with capacities on the nodes. However, all results can easily be applied to flow problems with capacities on the arcs (by a straightforward graph transformation). The main results in [23] are that the problem of finding a minimum congestion d-furcated flows is maxSNP-hard for fixed d and that congestion of a d-furcated flow is at most $1 + \frac{1}{d-1}$ times the congestion of a fractional flow. The definition of maxSNP-hard can be found in [76].

A closely related problem is the k-splittable flow problem [10], where the flow, for each commodity, is restricted to use at most k (not necessarily disjoint) paths. This problem came as a natural extension of the unsplittable flow problem [21, 57]. The unsplittable flow problem and the k-splittable flow problem are both NP-hard. It is well-known that a transportation problem can be equivalently recast into a flow problem (see Figure 6.3).

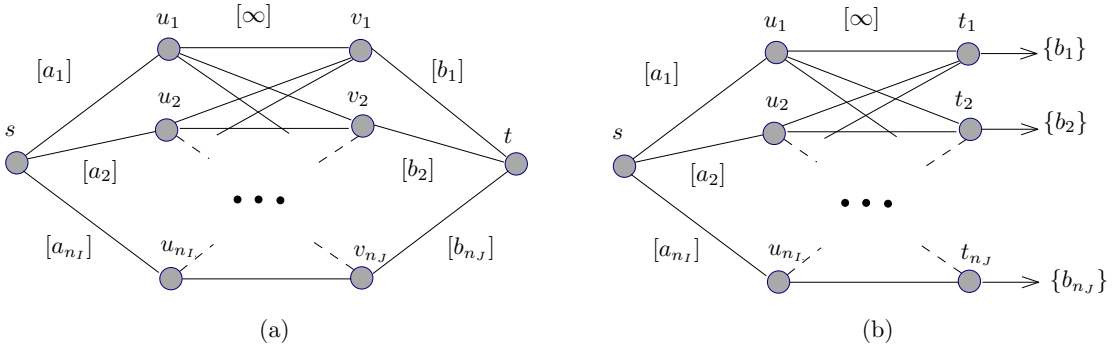


Figure 6.3: Transcription of a transportation problem (a) as a single commodity maximum (or minimum cost) $s-t$ flow, or (b) as a single source,multicommodity maximum concurrent flow. A symmetrical case of (b) would be a single destination multicommodity flow problem.

In fact, it turns out that d-FTP is a special case of d-furcated flow problem (using the flow model (a)) and d-CTP$_b$ is a special case of k-splittable flow problem (using the flow model (b)). It is interesting to note that, in this very particular type of graph (bipartite graph), bounding the number of path for a commodity is equivalent to bound the in-degree of the demand nodes (again, a symmetrical configuration is obtained where only the out-degree of supply nodes is bounded). The complexity results for d-furcated and k-splittable flows cannot hence be used as such to derive the complexity of the degree-bounded transportation problems.

To the best of our knowledge, the first paper addressing a bounded degree

transportation problem is [83] in which the problem, denoted $(1,3)$-FTP, where $d^+_{max} = 1$ and $d^-_{max} = 3$ is shown to be strongly NP-hard (using a reduction from 3-Partition). As observed in [8], the problem $(1,1)$-FTP is the classical assignment problem and problem (n_J, n_I)-FTP can be reduced to TP, both problems being known to be solvable in polynomial time. However, the case $(1,2)$-FTP is shown to be strongly NP-hard in [8] which also proposes a 1/2-approximation algorithm for all cases of (d^+_{max}, d^-_{max}).

In the problems we consider, the objective functions (obj_1), (obj_2) or (obj_3) come into play to discriminate among the feasible solutions of $\mathcal{S}(a,b,d)$. Since the problem of determining whether $\mathcal{S}(a,b,d)$ is empty or not is NP-hard, the complexity of our three problems follows.

Lemma 6.5.1. *Problems d-TP, d-MTP and d-BTP are NP-hard for $d = (1,2)$ and $d = (1,3)$.*

For other values of d (except $(1,1)$ and (n_I, n_J)), to the best of our knowledge, the complexity of these problems is still an open question.

6.6 Impact of the Degree Constraints

First note that the additional degree-constraints might not always be active. Indeed, in TP, the number of arcs in a simplex basis solution is $n-1$ and hence the average degree in such solutions is $2(n-1)/n < 2$. Since the objective function in the Minimax and Bottleneck Transportation problems is a threshold value, it has been shown that these problems can be solved in a parametric way, by fixing iteratively the values of the threshold. As a result, solutions of MTP and BTP can still be expressed using the transportation model structure, with an additional upper-bound on the flow variables. So, we recommend to solve all types of the degree constrained transportation problems by first solving the corresponding problems without the degree constraints to optimality, and checking afterwards if the additional degree constraints are already satisfied or not. Since, in random instances, the problems will often reach the optimality as well as the degree constraints, in other words, the degree constraints sometimes are redundant.

More generally, the impact of the degree constraints can be very different. The example of Figure 6.4(a) shows a transportation problem that has the same number of supply and demand nodes. The supplies and demands for all corresponding nodes are uniformly equal to 1, moreover, the cost on each link (i,j) is set to 1.

The optimal value of TP in this case is $v^*(\text{TP}) = n$. This solution is also valid for any level of constraints on the degrees (since the solution is obtained as a perfect matching): $v^*((d\text{-TP}) = n$. The same conclusion holds for the bottleneck problem (and all its degree constrained variants), since all costs are equal. On the contrary, for the Maxmin problem, each level of (uniform) degree constraints leads to different solution values: $v^*(d\text{-BTP}) = 1/d$ and $v^*(\text{BTP}) = 1/n$.

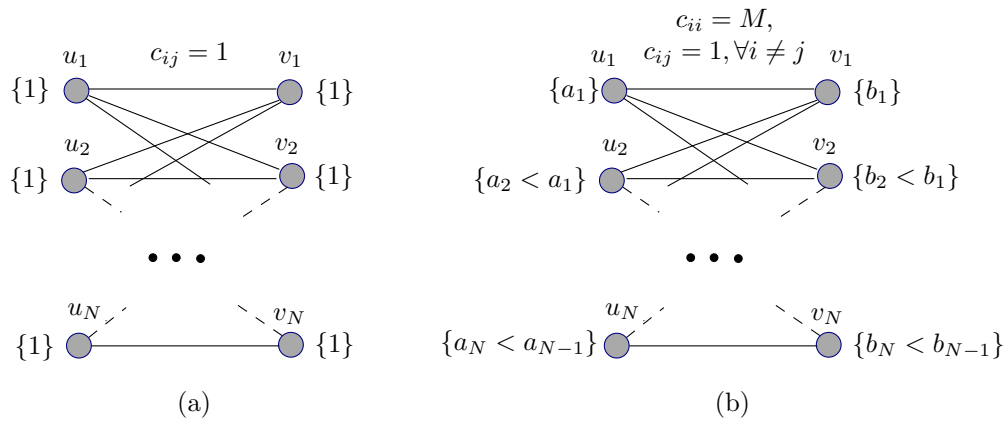


Figure 6.4: Two instances of transportation problem: (a) completely uniform case, $n_I = n_J = N$, all supplies and demands are equal to 1, all arc costs are equal to 1.

In another example in Figure 6.4(b), we still assume that the number of supply nodes is the same as the demand nodes, but the supplies and demands have the following relationship: $a_i = b_i, \forall i \in N$. The cost on each link follows the following rules: $\forall(i,j), c_{ij} = M, \text{if } i = j; c_{i,j} = 1, \text{if } li \neq j$. For each problem in this example, in terms of different objectives, when all degrees are bounded by 1 ($d = 1$), the only feasible solution to fit a supply with a demand is to take the horizontal arcs (with cost $M \gg 1$). Hence, the solutions for TP are: $v^*(\text{TP}) = n$ and $v^*(\text{1-TP}) = nM$. BTP and MTP have the same solutions when the degree bound is one: $v^*(\text{1-BTP}) = v^*(\text{1-BTP})M$. Without this tight degree constraint, the most costly arcs can be easily avoided: $v^*(\text{BTP}) = 1$ and $v^*(\text{MTP}) = 1/(n-1)$. This shows that, when a cost criterion is involved, the gap between 1-degree constrained and unconstrained problems optimal values is potentially unbounded (M can be chosen as large as one wishes).

6.7 Resolution Approaches

Since our problems contain, on the one hand, a "classical" transportation problem, and, on the other hand, an also "classical" b-matching problem, a reasonable

approach consists in trying to separate these two components, for instance, using a Lagrangian decomposition method. For this purpose, we will relax (in a Lagrangian fashion) the binding constraints (6.10). Consider for instance the degree constrained Transportation Problem d-TP:

$$d\text{-TP}\begin{cases} \min \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij}, & (6.20)\\ \sum_{j\in J} x_{ij} = a_i,\ \sum_{i\in I} x_{ij} = b_j, & \forall i\in I, j\in J, \quad (6.21)\\ 0 \le x_{ij} \le M_{ij}y_{ij}, & \forall i\in I, j\in J, \quad (6.22)\\ \sum_{j\in J} y_{ij} \le d^+(i),\ \sum_{i\in I} y_{ij} \le d^-(j), & \forall i\in I, j\in J, \quad (6.23)\\ y_{ij}\in\{0,1\}, & \forall i\in I, j\in J. \quad (6.24) \end{cases}$$

Denote by u_{ij} the Lagrangian variables associated with the binding constraints (6.22). Relaxing these binding constraints in a Lagrangian way, yields two parameterized subproblems:

$$SP_1(u): \quad z_1(u) = \min\left\{\sum_{i\in I}\sum_{j\in J}(c_{ij}+u_{ij})x_{ij} :\ (6.21),\ x_{ij}\ge 0\right\}, \quad (6.25)$$

$$SP_2(u): \quad z_2(u) = \max\left\{\sum_{i\in I}\sum_{j\in J} u_{ij}M_{ij}y_{ij} :\ (6.23),\ y_{ij}\in\{0,1\}\right\}. \quad (6.26)$$

The first one ($SP_1(u)$) is a standard transportation problem with costs $c_{ij} + u_{ij}$ on the arcs and the second one ($SP_2(u)$) is a standard maximum weight (bipartite) d-matching with weights $u_{ij}M_{ij}$ on the edges. Both problems can hence be solved very efficiently. If we denote $z(u) = z_1(u)+z_2(u)$, it is well known that $z(u) \le z^*(d\text{-TP}), \forall u \ge 0$ (in other words, the Lagrangian decomposition yields a relaxation of the problem). To achieve the best possible lower bound, we are hence interested in the problem:

$$\max\{z(u) :\ u \ge 0\}. \quad (6.27)$$

There are several "standard" ways to handle this dual problem. Note that, from a polyhedral point of view, the convex hull of all feasible solutions of d-TP, is the intersection of the TP polytope with the convex hull of the maximum weight matching polytope (which is known to be integral because the associated matrix is Totally Unimodular (TU) which has been defined in Theorem 3.3.1). However, most extreme points in this intersection have non-integer coordinates for the y vector.

Many other possibles approaches can be considered, for instance starting with greedy approaches or with the standard algorithms for transportation problems or maximum flows.

6.8 Conclusions and Future Work

In this chapter, we have proposed some new variants of degree constrained transportation problem and have showed that they are mainly related to two families of well-studied problems, namely, classical transportation problems and matching problems in bipartite graphs (which are also already related). The problems we consider include minmax or bottleneck components, which make the problems often harder to solve in practice. From a complexity point of view, although each individual component of our problems is polynomialy solvable, the combined problems are NP-hard. One promising line of research probably lies in the very particular structure of the problem polytope.

Chapter 7

Conclusions and Perspectives

Do not fear mistakes. You will know failure. Continue to reach out.

-Benjamin Franklin (1706-1790)

7.1 Conclusions

After the initial introduction of network coding [5], during the last decade this technique has been proposed as a powerful tools to improve telecommunication network from different aspects. Network coding provides a special perspective in electronic services delivery and digital distribution. It allows combining information in intermediate nodes and lets the coded information be decoded in those intermediate nodes or at terminals. This property is completely different from the traditional transportation of any kind but by recognizing the algebraic nature of digital commodities the new transportation method can be easily performed in the theory of information and potential applications in electronic devices and telecommunication networks. This method opens a new dimension to control data collision, in such a way that corresponding network performance may also change and needs to be revisited. Many studies have focused on the coding advantage from several aspects, such as throughput gain, reliability, and robustness, in different network settings, for example, multicast networks, wireless networks, and also distributed storage systems. The studies, for instance, in [20] and [52], have shown very convincing potential coding advantage in wireless networks and distributed storage systems. However, the initial advantage

claimed for network coding, namely throughput gain in multicast networks, is still relatively unclear in the literature. This is why in this thesis, we reconsider the evaluation of the throughput gain, focusing especially on multicast service; we evaluate the performance in both network coding strategy and traditional multicast routing. Based on the evaluation results, we propose a so-called strategic network coding which aims to reduce side-effects when applying network coding.

In Chapter 2, we give a review of network coding, which aims at helping the readers, who are not familiar with network coding, to understand the rest of thesis.

The difficulty for evaluating the throughput performance between network coding and multicast comes from finding the optimal throughput with multicast tree(s). The problem of finding network coding throughput is simpler, and it is polynomial time solvable. We first introduce some state-of-the-art algorithms as well as new polynomial time algorithms we created and which find the optimal end-to-end throughput when using single multicast trees. The extensive variety of algorithms can be accordingly applied in different types of graphs (undirected, directed and bidirected graphs). The algorithms proposed in Chapter 3, Section 3.5 and 3.6, are designed to solve the problem in the scenario where terminals are required to be on the leaves of the final tree.

In Chapter 4, we study a state-of-the-art mathematical model of the fractional Steiner tree packing problem. We confirm that the same formulations can be used to handle the problem in solving optimal end-to-end throughput when multiple multicast trees are used. Although the problem is NP-hard, we can solve it very efficiently when using column generation method, in some medium size random generated graphs. We verify on numerous sets of significantly large randomly generated instances that network coding does not outperform multicast when there is no limit on how many multicast trees are used. However, multicast usually requires many trees to approach the network coding throughput, and managing a great amount of multicast sessions in a network is unrealistic. Therefore, from a network management standpoint, network coding is a better solution when high throughput is sought. Moreover, we provide a heuristic algorithm which is based on column generation to calculate multicast throughput within limited number of trees. We find that with small number of trees, the multicast throughput is poor, but it is manageable this time. However, the throughput gain becomes obvious when comparing network coding strategy with multicast using limited number of trees. We provide another mathematical model in order to find a graph that has

maximum throughput gap between network coding and multicast. Solving this model is not trivial, so we merely test it on small size graphs. Surprisingly, we find that all the network topologies with maximum throughput gap are butterfly-like networks.

Although we confirm in Chapter 4 that network coding shows benefits in multicast services, the side effects it brings, such as additional overhead on coded packets, encoding and decoding cost during transmission, should not be overlooked. Based on the observations from Chapter 4, we examine two strategic network coding strategies in Chapter 5: one that limits the number of coding nodes and one that mixes multicast routing with limited number of trees and network coding. From the numerical results we obtain, we find that, in randomly generated networks, very few nodes are needed to perform encoding operations. In many cases, it is already sufficient to achieve network coding throughput when only the source acts as an encoding node. We then focus on the strategy that mixes multicast routing and network coding. This strategy aims to introduce a small amount of network coding flow in order to help increase the throughput when we apply limited number of multicast trees. The benefits of this strategy are twofold: it reduces the side effects as compared to the scenarios where pure network coding flows are used; it increases the network throughput as compared to the cases where we apply multicast routing with limited trees. However, in order to approach the network coding throughput by using our mixed strategy, a significant part of multicast traffic must be processed by network coding, which indicates that network coding is still prominent in achieving a better multicast throughput.

In Chapter 6, we investigate a degree constrained transportation problem which has been rarely studied in the literature from the perspective on a routing problem in a coded distributed storage system. To the best of our knowledge, there is no algorithm that solves the general cases of this problem, only approximation algorithms have been proposed. We explain the relationship between this special transportation problem and the two classical combinatorial problems, transportation problem and matching problem. We derive some feasibility conditions for this particular problem and clarify the complexity issues for some general cases. In addition, we propose a resolution approach based on Lagrangian decomposition. We will further investigate this approach in future research, since so far the convergence of this algorithm is slow.

7.2 Perspectives

There are two major perspectives that we wish to pursue after the works contained in this dissertation.

The evaluation of the coding advantage in Chapter 4 focuses on scenarios which have single multicast connection. A multicast connection is used to delivery one media stream or one generation of a content. But, in real networks, there may be multiple connections at the same time, especially for content delivery. There may be several generations sent simultaneously from multiple sources to terminals. It is worth to evaluate the coding advantage for these scenarios of multiple multicast connections as well. We can also extend the research on coding advantage, in terms of network survivability, reliability, robustness and network security, in multicast services.

In order to have some insight on the coding advantage from an economical point of view, we can also consider some cost with network use, then compare throughput gain among different multicast strategies which are shown in this dissertation under certain cost threshold. This line of research can be classified into a so-called techno-economical analysis which can greatly assist in averting misspent efforts and help future investment. To the best of our knowledge, this kind of research in network coding field is not common but extremely valuable for most actors in the telecommunication industry.

Publications

- Eric Gourdin and Yuhui Wang, **Some further investigation on maximum throughput: Does network coding really help?**, In *Proceedings of the 24th International Teletraffic Congress (ITC)*, 2012.

- Eric Gourdin and Yuhui Wang, **Bottleneck and Maxmin Transportation Problems with Degree Constraints**, In *Local Proceedings of the 2nd International Symposium on Combinatorial Optimization (ISCO)*, 2012.

- Eric Gourdin, Yuhui Wang and Muriel Médard, **Strategic Network Coding - How much and Where to Code to Obtain Most of the Benefits**, accepted by *IEEE International Symposium on Network Coding*, 2013.

- Eric Gourdin and Yuhui Wang, **Efficient Algorithms for Bottleneck Steiner Tree Problems**, submitted.

- Eric Gourdin and Yuhui Wang, **Optimization Models for a Techno-economical Analysis of Network Coding Advantage**, in preparation.

Bibliography

[1] ARD Digital IPTV. http://www.ard-digital.de/index.php?id=14026&languageid=1. [Online; accessed 08-January-2013].

[2] BBC Multicast Home. http://www.bbc.co.uk/multicast/radio/. [Online; accessed 08-January-2013].

[3] Cisco Visual Networking Index: Forecast and Methodology, 2011-2016. http://www.cisco.com/en/US/netsol/ns827/networking_solutions_white_papers_list.html, May 2012. [Online; accessed 15-November-2012].

[4] S. Acedański, S. Deb, M. Médard, and R. Koetter. How Good is Random Linear Coding based Distributed Networked Storage. In *NetCod*, 2005.

[5] R. Ahlswede, N. Cai, S.-Y.R. Li, and R.W. Yeung. Network Information Flow. *IEEE Transactions on Information Theory*, 46(4):1204 –1216, July 2000.

[6] R. K. Ahuja and and J. B. Orlin T. L. Magnanti. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall Englewood Cliffs, 1993.

[7] R.K. Ahuja. Algorithms for the Minimax Transportation Problem. *Naval Research Logistics Quaterly*, 33:725–739, 1986.

[8] E. Akçali and A. Üngör. Approximation Algorithms for Degree-Constrained Bipartite Network flow. In *ISCIS*, pages 163–170. Springer.

[9] G. Baier, E. Köhler, and M. Skutella. On the k-Splittable Flow Problem. In *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 101–113, 2002.

[10] G. Baier, E. Köhler, and M. Skutella. The k-Splittable Flow Problem. *Algorithmica*, 42:231–248, 2005.

[11] J. Barros, R.A. Costa, D. Munaretto, and J. Widmer. Effective Delay Control for Online Network Coding. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pages 208 – 216, April 2009.

[12] P.M. Camerini. The Min-Max Spanning Tree Problem and Some Extensions. *Inform. Processing Letters*, 7(1):10–14, January, 1978.

[13] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-Bandwidth Multicast in Cooperative Environments. In *Proceedings of of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[14] D-S. Chen, R. G. Baston, and Y. Dang. *Applied Integer Programming: Modelling and Solution*. John Wiley & Sons, Inc., 2010.

[15] Y. H. Chen, C. L. Lu, and C. Y. Tang. On the Full and Bottleneck Full Steiner Tree Problems. In *Proceedings of the 9th Annual International Conference (COCOON)*, pages 25–28, 2003.

[16] C. Chiang, M. Sarrafzadeh, and C.K. Wong. Global Routing Based on Steiner Min-Max Trees. *IEEE Transaction on Computer-Aided Design*, 9(12):1318–1325, December 1990.

[17] N. Cleju, N. Thomos, and P. Frossard. Selection of Network Coding Nodes for Minimal Playback Delay in Streaming Overlays. *IEEE Transactions on Multimedia*, 13(5):1204 –1216, November, 2011.

[18] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[19] U. Derigs and U. Zimmermann. An Augmenting Path Method for Solving the Linear Bottleneck Transportation Problems. *Computing*, 22:1–15, 1979.

[20] A.G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A Survey on Network Codes for Distributed Storage. In *Proceedings of the IEEE*, pages 476 –489, March, 2011.

[21] Y. Dinitz, N. Garg, and M.X. Goemans. On the Single-Source Unsplittable Flow Problem. In *Proceedings of 39th Annual Symposium on Foundations of Computer Science*, pages 290 –299, November, 1998.

[22] P. Donovan, B. Shepherd, A. Vetta, and G. Wilfong. Degree-Constrained Network Flows. In *Proceedings of the 39th annual ACM symposium on Theory of Computing*, pages 681–688, 2007.

[23] P. Donovan, B. Shepherd, A. Vetta, and G. Wilfong. Degree-Constrained Network Flows. In *Proceedings of the 39th annual ACM symposium on Theory of Computing*, pages 681–688, 2007.

[24] C.W. Duin and A. Volgenant. The Partial Sum Criterion for Steiner Trees in Graphs and Shortest Paths. *European Journal of Operations Research*, 97:172–182, 1997.

[25] J. Edmonds. Edge-Disjoint Branchings. *Combinatorial Algorithms, Algorithmics Press, New York*, pages 91–96, 1972.

[26] J. Edmonds and R. M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM*, 19(2):248–264, April 1972.

[27] U. J. Ferner, M. Médard, and E. Soljanin. Toward Sustainable Networking: Storage Area Networks with Network Coding. *CoRR*, May 2012.

[28] Jr. L. R. Ford and D. R. Fulkerson. Maximum Flow through a Network. *Canadian Journal of Mathematics*, 8:399 – 404, 1956.

[29] Jr. L. R. Ford and D. R. Fulkerson. A Simple Algorithm for Finding Maximal Network Flows and an Application to the Hitchcock Problem. *Canadian Journal of Mathematics*, 9:210 –218, 1957.

[30] Jr. L. R. Ford and D. R. Fulkerson. A Suggested Computation for Maximal Multi-commodity Network Flows. *Manage. Sci*, 5:97 –101, July 1958.

[31] M. L. Fredman and R. E. Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *J. ACM*, 34(3):596–615, 1987.

[32] H. N. Gabow and R. E. Tarjan. Algorithms for Two Bottleneck Optimization Problems. *Journal of Algorithms*, 9:411–417, 1988.

[33] R. S. Garfinkel and M. R. Rao. The Bottleneck Transportation Problem. *Naval Research Logistics Quaterly*, 18:465–472, 1971.

[34] N. Garg, R. Khandekar, K. Kunal, and V. Pandit. Bandwidth Maximization in Multicasting. In *Proceedings of European Symposium on Algorithms*, pages 242–253, 2003.

[35] L. Georgiadis. Bottleneck Multicast Trees in Linear Time. *IEEE Commun. Lett.*, September 2003.

[36] C. Gkantsidis and P. R. Rodriguez. Network Coding for Large Scale Content Distribution. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, volume 4, pages 2235 – 2245, March 2005.

[37] M. X. Goemans and Y. S. Myung. A Catalog of Steiner Tree Formulation. *Neworks*, 23:19–28, 1993.

[38] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire. Femtocaching: Wireless Video Content Delivery through Distributed Caching Helpers. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pages 1107 –1115, March 2012.

[39] E. Gourdin and Y. Wang. Some Further Investigation on Maximum Throughput: Does Network Coding Really Help? In *Proceedings of the 24th International Teletraffic Congress (ITC)*, 2012.

[40] P. L. Hammer. Time Minimizing Transportation Problems. *Naval Research Logistics Quaterly*, 16:345–367, 1969.

[41] T. Ho and D. S. Lun. *Network Coding an introduction*. Cambridge University Press, New York, USA, 2008.

[42] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong. Toward a Random Operation of Networks. *IEEE Transactions on Information Theory*, pages 1–8, 2004.

[43] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong. A Random Linear Network Coding Approach to Multicast. *IEEE Transactions on Information Theory*, 52(10):4413 –4430, October 2006.

[44] T. Ho, M. Médard, J. Shi, M. Effros, and D. R. Karger. On Randomized Network Coding. In *Proceedings of 41st Annual Allerton Conference on Communication, Control, and Computing*, 2003.

[45] D. S. Hochbaum and G. J. Woeginger. A Linear-Time Algorithm for the Bottleneck Transportation Problem with a Fixed Number of Sources. *Operations Research Letters*, 24:25–28, 1999.

[46] S. Jaggi, P. Sanders, P. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen. Polynomial Time Algorithms for Multicast Network Code Construction. *IEEE Transactions on Information Theory*, 51(6):1973 – 1982, June 2005.

[47] K. Jain, M. Mahdian, and M. R. Salavatipour. Packing Steiner Trees. In *Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms*, pages 266–274, 2003.

[48] V. Kaibel and M. A.F. Peinhardt. On the Bottleneck Shortest Path Problem. Technical report, 2001.

[49] S. Katti, S. Gollakota, H. Balakrishnan, and M. Médard. Symbol-Level Network Coding for Wireless Mesh Networks. In *Proceedings of ACM SIGCOMM*, pages 401 – 412, August 2008.

[50] S. Katti, S. Gollakota, and D. Katabi. Embracing Wireless Interference: Analog Network Coding. In *Proceedings of ACM SIGCOMM*, pages 397 – 408, August 2007.

[51] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Growcroft. Xors in the Air: Practical Wireless Network Coding. In *Proceedings of ACM SIGCOMM*, September 2006.

[52] M. Kim. *Network Coding for Robust Wireless Networks*. PhD thesis, Massachusetts Institute of Technology, 2012.

[53] M. Kim, C. W. Ahn, M. Médard, and M. Effros. On Minimizing Network Coding Resources: An Evolutionary Approach. In *NetCod*, 2006.

[54] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. J. Leith, and M. Médard. Network Coded TCP (CTCP). *CoRR*, 2012.

[55] M. Kim, M. Médard, V. Aggarwal, U. M. O'Reilly, W. Kim, C. W. Ahn, and M. Effros. Evolutionary Approaches to Minimizing Network Coding Resources. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pages 1991 – 1999, May 2007.

[56] M. Kim, M. Médard, and J. Barros. Modeling Network Coded TCP Throughput: A Simple Model and its Validation. In *Proceedings of International ICST/ACM Conference on Performance Evaluation Methodologies and Tools (Valuetools)*, May 2011.

[57] J. M. Kleinberg. Single-Source Unsplittable Flow. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 68–77, 1996.

[58] T. Koch and A. Martin. Solving Steiner Tree Problems in Graphs to Optimality. *Networks*, 32:207–232, 1998.

[59] R. Koetter and M. Médard. An Algebraic Approach to Network Coding. *IEEE/ACM Transactions on Networking*, 11(5):782 – 795, October 2003.

[60] V. P. Kompella, G. C. Polyzos, and J. C. Pasquale. Multicast Routing for Multimedia Communication. *IEEE/ACM Transactions on Networking*, 1(3):286–292, 1993.

[61] A. Kulkarni, M. Heindlmaier, D. Traskov, M. J Montpetit, and M. Médard. An Implementation of Network Coding with Association Policies in Heterogeneous Networks. In *Proceedings of the IFIP TC 6th international conference on Networking*, pages 110–118, 2011.

[62] D. Leong, A. G. Dimakis, and T. Ho. Distributed Storage Allocation Problems. In *NetCod '09 Workshop on Network Coding, Theory, and Applications*, pages 86 – 91, June 2009.

[63] D. Leong, A. G. Dimakis, and T. Ho. Distributed Storage Allocation for High reliability. In *IEEE International Conference on Communications (ICC)*, pages 1 – 6, May 2010.

[64] D. Leong, A. G. Dimakis, and T. Ho. Distributed Storage Allocations. *IEEE Transactions on Information Theory*, 58(7):4733 –4752, July 2012.

[65] S. Y R. Li, R.W. Yeung, and C. Ning. Linear Network Coding. *IEEE Transactions on Information Theory*, 49(2):371 –381, February 2003.

[66] Z. Li, B. Li, D. Jiang, and L. C. Lau. On Achieving Optimal Throughput with Network Coding. In *Proceedings of IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 2184 – 2194, March 2005.

[67] Z. Li, B. Li, and L. C. Lau. A Constant Bound on Throughput Improvement of Multicast Network Noding in Undirected Networks. *IEEE Transactions on Information Theory*, 55(3):1016 – 1026, March 2009.

[68] D. E. Lucani and M. Médard. Bridging Tree-Packing and Network Coding: An Information Flow Approach. In *Proceedings of 45th Conference of Information Sciences and Systems (CISS)*, pages 1 – 6, 2011.

[69] D. S. Lun, Muriel Médard, T. Ho, and R. Koetter. Network Coding with a Cost Criterion. In *Proceedings of 2004 International Symposium on Information Theory and its Applications (ISITA)*, pages 1232–1237, 2004.

[70] D. S. Lun, N. Ratnakar, R. Koetter, M. Médard, E. Ahmed, and H. Lee. Achieving Minimum-Cost Multicast: A Decentralized Approach Based on Network Coding. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pages 1607–1617, 2005.

[71] D. S. Lun, N. Ratnakar, M. Médard, R. Koetter, D. R. Karger, T. Ho, E. Ahmed, and F. Zhao. Minimum-Cost Multicast over Coded Packet Networks. *IEEE/ACM Transactions on Networking*, 14:2608–2623, 2006.

[72] M. Montpetit and M. Médard. Video-Centric Network Coding Strategies for 4G Wireless Networks: An Overview. In *Proceedings of 7th IEEE Consumer Communications and Networking Conference (CCNC)*, pages 1 –5, January 2010.

[73] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., New York, USA, 1988.

[74] Z. Nutov. Approximating Directed Weighted-Degree Constrained Networks. In *Proceedings of the 11th International Workshop APPROX and 12th International Workshop RANDOM on Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques*, pages 219–232, 2008.

[75] J. B. Orlin. A Faster Strongly Polynomial Minimum Cost Flow Algorithm. In *Proceedings of the 20th Annual ACM Symposium on Computing Theory*, pages 355–362, 1988.

[76] C. Papadimitriou and M. Yannakakis. Optimization, Approximation, and Complexity Classes. In *Proceedings of the 20th annual ACM symposium on Theory of Computing*, pages 229–234, 1988.

[77] M. G. C. Resende and P. M. Pardalos. *Handbook of Opertimization in Telecommunications*. Spinger Science and Business Media, New York, USA, February 2006.

[78] G. Robins and A. Zelikovsky. Improved Steiner Tree Approximation in Graphs. *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 770–779, 2000.

[79] L. H. Sahasrabuddhe and B. Mukherjee. Multicast Routing Algorithms and Protocols: A Tutorial. *IEEE Networks*, 14:90–102, 2000.

[80] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York, USA, 1986.

[81] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer-Verlag, Berlin Heidelberg, 2003.

[82] J. K. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. Barros. Network Coding Meets TCP: Theory and Implementation. pages 280 – 288, April 2009.

[83] L. B. Toktay and R. Uzsoy. A Capacity Allocation Problem with Integer Side Constraints. *European Journal of Operational Research*, 109(1):170–182, 1998.

[84] H. Wang, J. Liang, and C. C. J. Kuo. Overview of Robust Video Streaming with Network Coding. *Journal of Visual Communication and Image Representation*, 2010.

[85] B. M. Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[86] Y. Wu, P. A. Chou, and K. Jain. A Comparison of Network Coding and Tree Packing. In *Proceedings of International Symposium on Information Theory (ISIT)*, page 143, 2004.

[87] X. Yin, X. Wang, J. Zhao, X. Xue, and Z. Li. On Benefits of Network Coding in Bidirected Networks and Hyper-Networks. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pages 325 – 333, March 2012.