Optimization of information flows in telecommunication networks

Thibaut Lefebvre

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS | le cnam

École Doctorale d'Informatique Télécommunications et Électronique

Laboratoire CEDRIC, équipe Optimisation Combinatoire

THÈSE DE DOCTORAT

présentée par : **Thibaut LEFEBVRE**

soutenue le : **27 juin 2016**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline / Spécialité : **Informatique / Recherche Opérationnelle**

Optimization of information flows in telecommunication networks

THÈSE DIRIGÉE PAR

Mme ELLOUMI Sourour	*MCF HDR, ENSIIE*
M. BENTZ Cédric	*MCF, CNAM*
M. GOURDIN Éric	*Ingénieur de recherche, Orange*

RAPPORTEURS

M. FORTZ Bernard	*PR, Université Libre de Bruxelles*
M. NACE Dritan	*PR, Université de Technologie de Compiègne*

EXAMINATEURS

Mme BAZGAN Cristina	*PR, Université Paris Dauphine*
M. BEN-AMEUR Walid	*PR, TELECOM SudParis*
M. PICOULEAU Christophe	*PR, CNAM*

À mes parents

If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is.

John von Neumann

Remerciements

Je tiens à remercier ici les nombreuses personnes qui ont contribué, de près ou de loin, à l'aboutissement de ma thèse.

Je souhaite tout d'abord remercier ma directrice de thèse, Sourour Elloumi, pour sa présence à mes côtés et son soutien tout au long du périple et pour m'avoir transmis la passion et les valeurs de la recherche académique. Je tiens également à remercier Éric Gourdin, d'abord pour m'avoir offert l'opportunité de passer trois ans au sein d'Orange, ensuite pour l'encadrement, le soutien et la protection qu'il m'a prodigués afin que je puisse effectuer mes travaux dans d'excellentes conditions. Enfin, je désire remercier Cédric Bentz pour avoir accepté d'encadrer ma thèse, ainsi que pour les longues heures que nous avons passées à étudier des problèmes variés de théorie de la complexité. Finalement, je souhaite remercier collectivement mes trois encadrants, d'abord pour m'avoir supporté (aux deux sens du terme) pendant ces années, ensuite pour le savant mélange de liberté et d'orientation qu'ils ont su me prodiguer afin de me guider dans mon parcours, enfin pour la patience et le dévouement dont ils ont fait preuve lors de la phase de rédaction du présent manuscrit.

Je remercie les Professeurs Bernard Fortz et Dritan Nace d'avoir accepté d'être rapporteurs de ma thèse. Je tiens également à remercier les Professeurs Cristina Bazgan, Walid Ben-Ameur et Christophe Picouleau pour avoir accepté de faire partie de mon jury de thèse.

Je tiens ensuite à remercier la société Orange pour le financement de mes travaux de recherche pendant les trois années de mon contrat avec le groupe.

Je tiens à remercier chaleureusement Maurice pour toute l'aide qu'il m'a apportée et les conseils qu'il m'a prodigués afin que je puisse mener à bien la phase "expérimentation" de ma thèse. Je lui suis reconnaissant de m'avoir formé aux langages Ruby et Julia ainsi qu'aux bonnes pratiques du développement logiciel. Merci aussi à son collègue de bureau Christophe pour son enthousiasme communicatif (surtout lors de réunion à l'improviste dans un RER).

Je souhaite maintenant remercier toutes les personnes que j'ai eu l'opportunité de côtoyer pendant mes trois années à Orange. Je pense en particulier à Nabil pour les nombreuses opportunités qu'il m'a données de présenter mes travaux de recherche dans des conférences internationales. Je pense aussi à Florence pour son excellent cours de théorie de l'information, à Ruby pour nos échanges autour de l'utilisation du codage dans les centres de données, ou encore à Adam pour l'organisation de séminaires sur l'apprentissage automatique. Je tiens également à remercier Régine pour son aide lors de la mise en place du cadre légal de ma thèse. Je remercie aussi tous les membres de l'équipe TRM pour leur accueil et leur soutien pendant ces trois ans, notamment Pierre, Amal, Mathieu, Yannick, Claudio, Raluca, Bruno, Wuyang, Luca, Philippe, Félipe, Nancy, Alain, Christian, Bobby et Yuhui. J'ai une pensée particulière pour tous les doctorants encore attelés à la tâche au moment où j'écris ces lignes : Paul, Guanglei, Léonce et Yassine. Je souhaite également remercier les personnes que j'ai eu la chance de croiser lors des séminaires de RO régulièrement organisés par Éric : Vincent, Matthieu, Prosper, Cédric et Stéphane.

Ma reconnaissance va également aux membres du laboratoire CEDRIC du CNAM. Merci à Anne d'avoir accepté d'être membre de mon jury de mi-parcours. Merci à Viviane et à Stéphane pour nos conversations enrichissantes. Merci à Sami d'avoir toléré ma présence dans son bureau. Merci en particulier aux membres de l'équipe OC pour leur soutien et la motivation qu'ils m'ont communiqués. Je pense notamment à Amélie, Agnès et Daniel. Merci aussi aux membres du bureau des doctorants avec qui j'ai eu l'occasion de discuter lors de mes passages au CNAM : Pierre-Louis, Sabine et Dimitri. Une pensée pour Thomas qui poursuit sa thèse au moment de la rédaction de ces lignes. Merci à Alain Billionnet et à toute l'équipe RO de l'IIE pour m'avoir donné le goût de cette discipline. Merci à Alain Faye pour m'avoir offert l'opportunité de transmettre ce goût aux nouveaux étudiants de

l'école. Un grand merci à Marie-Christine pour sa contribution à la fois au commencement et à l'achèvement de mon doctorat.

Je ne peux pas manquer non plus de remercier tous mes anciens collègues d'EURODECISION qui m'ont soutenu lors de la dernière phase de rédaction de mon manuscrit, en particulier Edwin, Salif, Cheikh, Philippe, et Romain. Enfin, merci à Denis et à Céline pour m'avoir offert l'opportunité d'achever mon manuscrit.

Merci à David, Haïtem et Gen pour les expéditions "héroïques" sur GW2. Merci à Maxime et Stanislas pour leur soutien. Merci à Marion pour ses encouragements. Un grand merci à Grégoire qui m'a constamment balisé le chemin depuis le Master.

Finalement, je tiens à remercier ma famille pour son soutien indéfectible tout au long de ma thèse. Merci à Vanda pour sa foi en moi et un immense merci à mes parents pour leur amour.

Résumé

Dans les réseaux de télécommunications, la demande croissante pour de nouveaux services, comme la diffusion de vidéos en continu ou les conférences en ligne, engendre un besoin pour des dispositifs de télécommunication où le même contenu est acheminé depuis un émetteur unique vers un groupe de récepteurs. Cette évolution ouvre la voie au développement de nouvelles techniques d'acheminement des données, comme le multicast qui laisse un nœud du réseau copier ses données d'entrée puis retransmettre ces copies, ou le codage réseau, qui est une technique permettant à un nœud d'effectuer des opérations de codage à partir de ses données d'entrée. Cette thèse traite de la mise en place de techniques de codage au sein d'un réseau multicast filaire. Nous formalisons certains problèmes qui apparaissent naturellement dans ce contexte grâce à la recherche opérationnelle et à des outils d'optimisation mathématique. Notre objectif est de développer des modèles et des algorithmes afin de calculer, au moins de manière approchée, certaines grandeurs qui ont vocation à être pertinentes dans le cadre de la comparaison de techniques d'acheminement de données dans un réseau de télécommunications. Nous évaluons ainsi, d'un point de vue à la fois théorique et expérimental, l'impact induit par l'introduction de techniques de codage au sein d'un réseau multicast. Nous nous concentrons en particulier sur des critères importants pour un opérateur de télécommunication, comme la maximisation du débit d'information entre une source et un ensemble de destinataires dans le réseau, la minimisation de la congestion sous contrainte de demande, ou la minimisation de la perte de débit ou du coût induit par l'acheminement des données dans un réseau soumis à des pannes.

Mots clés : Optimisation réseau, Programmation mathématique, Codage réseau, Multicast, Multiflot, Arbre de Steiner

Abstract

In telecommunication networks, the increasing demand for new services, like video-streaming or teleconferencing, along with the now common situation where the same content is simultaneously requested by a huge number of users, stress the need for point to many data transmission protocols where one sender wishes to transmit the same data to a set of receivers. This evolution leads to the development of new routing techniques like multicast, where any node of the network can copy its received data and then send these copies, or network coding, which is a technique allowing any node to perform coding operations on its data. This thesis deals with the implementation of coding techniques in a wired multicast network. We formalize some problems naturally arising in this setting by using operations research and mathematical optimization tools. Our objective is to develop models and algorithms which could compute, at least approximately, some quantities whose purpose is to be relevant as far as forwarding data using either multicast and network coding in telecommunications networks is concerned. We hence evaluate, both in theory and numerically, the impact of introducing coding techniques in a multicast network. We specifically investigate relevant criteria, with respect to the field of telecommunications, like the maximum amount of information one can expect to convey from a source to a set of receivers through the network, the minimum congestion one can guarantee while satisfying a given demand, or the minimum loss in throughput or cost induced by a survivable routing in a network prone to failures.

Keywords: Network optimization, Mathematical programming, Network coding, Multicast, Multicommodity flow, Steiner tree

Résumé des travaux de thèse

Introduction

Motivation

L'émergence d'Internet et des réseaux de télécommunications modernes constitue un changement majeur dans l'histoire humaine. Ces réseaux de télécommunications ont rendu possible la transmission d'informations de manière instantanée et fiable d'un point à un autre de la planète. Il s'en est suivi une révolution dans les modes de communications et la manière dont nous interagissons avec notre environnement. Il est ainsi loisible à tout un chacun de communiquer en temps réel avec une personne (ou une machine) située dans l'autre hémisphère. De plus, bien que l'intention à l'origine de la plupart de nos actions soit l'obtention d'une modification de notre environnement proche, il n'en demeure pas moins que l'impact de ces mêmes actions peut acquérir une portée planétaire. Ainsi, un phénomène initialement local peut acquérir une dimension globale de par sa propagation au sein d'un réseau de télécommunications, ce dernier engendrant un effet de rétroaction positive (au sens cybernétique du terme).

D'un point de vue historique, les premiers dispositifs de télécommunications, comme les pigeons voyageurs, le télégraphe ou le téléphone, ont été développés pour permettre la communication d'un point à un autre du réseau : Un émetteur souhaite expédier un message à un récepteur unique au sein du réseau. Dans une certaine mesure, Internet a lui aussi été pensé dans cette optique d'utilisation. De nos jours, la demande croissante pour de nouveaux types de services, comme la diffusion de vidéos en continu ou les conférences en ligne, engendre un besoin pour des dispositifs de télécommunications ou le même contenu est acheminé depuis un émetteur unique vers un groupe de récepteurs. Cette évolution

dans l'utilisation des réseaux de télécommunications ouvre la voie au développement de nouveaux services afin de répondre à ces besoins. D'un point de vue technique, il s'avère cependant nécessaire d'adapter les réseaux afin d'être en mesure de répondre à ce nouveau type de requête. Une manière de le faire consiste à modifier le mécanisme de diffusion de l'information dans un réseau de télécommunications filaire qui repose sur la transmission de paquets de données. Une voie prometteuse consiste à effectuer d'autres opérations que la seule transmission de paquets au niveau des nœuds du réseau de télécommunication.

Nous allons à présent décrire succinctement le principe de fonctionnement d'un réseau de télécommunications filaire avant de présenter les stratégies alternatives d'acheminement de l'information au sein d'un réseau étudiées dans le présent manuscrit.

Objectifs

Les travaux effectués dans cette thèse ont été conduits avec deux objectifs principaux. D'abord, nous cherchons à formaliser, et si possible à résoudre, certains problèmes qui apparaissent naturellement dans le domaine de l'acheminement d'informations dans un réseau de télécommunications à l'aide d'outils issue de l'optimisation mathématique et plus particulièrement de l'optimisation dans les réseaux. Ensuite, nous proposons des modèles et des algorithmes afin de pouvoir évaluer, d'un point de vue tant théorique que pratique, l'impact de certaines techniques d'acheminement de l'information sur le fonctionnement d'un réseau de télécommunications. Plus précisément, nous cherchons à déterminer l'impact de la mise en place de *techniques de codage* au sein d'un réseau dont les nœuds sont déjà équipés d'un dispositif de diffusion *multicast* (les définitions sont données dans la section suivante).

Nous concentrons notre attention sur les aspects combinatoires des problèmes que nous étudions ce qui nous pousse à ne pas considérer d'autres aspects connexes et cruciaux de ces mêmes problèmes. Le présent manuscrit peut donc être considéré comme s'inscrivant dans le champ de l'optimisation dans les réseaux plutôt que dans celui des télécommunications, bien que les travaux présentés ici prennent leur source au sein de ce dernier domaine. Le souhait de l'auteur est que les chercheurs et les praticiens des télécommunications trouvent dans le présent document des idées et des outils utiles à la résolution de leurs problèmes.

Sujet de la thèse

Vocabulaire de la théorie des graphes

Nous allons souvent représenter un réseau de télécommunications à l'aide d'un graphe, de sorte qu'il est opportun de donner d'emblée quelques notions de théorie des graphes qui nous seront utiles par la suite. Un *graphe non orienté* $G = (V, E)$ est la donnée d'un ensemble fini V dont les éléments sont appelés les *sommets* du graphe, et d'un sous-ensemble E de l'ensemble de toutes les paires de sommets distincts, dont les éléments sont appelés les *arêtes* du graphe. Un *graphe orienté* $D = (V, A)$ est la donnée d'un ensemble fini V dont les éléments sont également appelés sommets du graphe orienté, et d'un sous-ensemble A de l'ensemble de tous les couples de sommets distincts, dont les éléments sont appelés les *arcs* du graphe orienté. Enfin, un *graphe bi-orienté* $B = (V, L)$ est un cas particulier de graphe orienté tel que, si un couple de sommets (u, v) fait partie de l'ensemble des arcs de B, alors le couple de sommets (v, u) fait également partie de ce dernier ensemble. La paire d'arcs (u, v) et (v, u) est appelée un *lien*. Dans la notation $B = (V, L)$, L fait référence à l'ensemble des liens du graphe bi-orienté. Nous utilisons souvent le terme générique *graphe* pour englober les trois cas précédents. Tous les graphes que nous considérons sont supposés sans boucle, ont au moins deux sommets et au moins un lien, et comportent au plus un lien entre deux sommets donnés. L'*ordre* d'un graphe est son nombre de sommets tandis que sa *taille* fait référence à son nombre d'arêtes ou d'arcs (suivant le type de graphe considéré).

Un graphe $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ est un *sous-graphe* d'un graphe $D = (V, A)$, ce que l'on note $\mathcal{D} \subseteq D$, si $\mathcal{V}$ est un sous-ensemble de V et que $\mathcal{A}$ est un sous-ensemble de l'intersection de A avec l'ensemble des paires / couples de sommets distincts de $\mathcal{V}$. Étant donné un graphe $D = (V, A)$, un *chemin* p dans D est un sous-graphe de D dont les arcs peuvent être ordonnés de sorte à ce que l'origine de chaque arc de p coïncide avec l'extrémité de l'arc qui le précède dans l'ordre. Un *cycle* dans un graphe $D = (V, A)$ est un chemin dont l'extrémité du dernier arc dans l'ordre est l'origine du premier arc par rapport à l'ordre. Un graphe est *acyclique* s'il ne contient pas de cycle parmi ses sous-graphes. Un chemin acyclique est dit *simple*. Un graphe $D = (V, A)$ est *connecté du point de vue d'un sommet* v s'il existe (au moins) un chemin d'origine v vers chaque autre sommet du graphe D. Un

arbre enraciné en un sommet v dans un graphe $D = (V, A)$ est un sous-graphe acyclique de D contenant v et connecté du point de vue de v. Le lecteur consultera l'ouvrage de West et al. [1] et celui de Bondy et Murty [2] pour une introduction plus complète à la théorie des graphes. Concernant les graphes orientés, nous renvoyons le lecteur vers le livre de Bang-Jensen et Gutin [3]. Enfin, nous souhaitons mentionner l'ouvrage de Chartrand et al. [4].

Modèles de réseaux

De manière informelle, un *réseau* est une collection de dispositifs, appelés *nœuds du réseau*, inter-connectés entre eux de sorte à ce qu'un sous-ensemble de nœuds, appelés *terminaux*, puissent communiquer entre eux. Un *nœud intermédiaire* est un nœud du réseau qui n'est ni la source, ni un terminal. La communication entre deux nœuds du réseau s'effectue par la transmission de *messages*. L'un des terminaux, appelé la *source*, dispose de l'ensemble des données que souhaitent obtenir tous les autres, les *terminaux*. Un nœud du réseau peut recevoir un message en provenance d'un autre nœud avant de lui-même transmettre ce message à un troisième nœud. Deux nœuds du réseau sont *connectés* s'il est possible à au moins l'un d'entre eux de directement transmettre un message à l'autre, sans passer par un troisième nœud au préalable.

Il est assez naturel de modéliser un réseau de télécommunications à l'aide d'un graphe (orienté ou non suivant les besoins) $D = (V, A)$ où chaque nœud est modélisé par un sommet et deux sommets sont reliés par un arc ou une arête s'ils sont connectés au sens défini précédemment. La structure du graphe résultant est fonction du type de transmission autorisé entre les nœuds. Si, pour chaque paire de nœuds connectés, les deux nœuds peuvent simultanément envoyer un message et en recevoir un en provenance de l'autre nœud, le graphe sera bi-orienté. Si au contraire l'un des deux nœuds de la paire doit jouer le rôle de récepteur tandis que l'autre joue le rôle d'émetteur, les rôles pouvant être échangés, le graphe associé est non orienté. Enfin, si le rôle de chaque nœud au sein d'une paire est figé, l'un est toujours l'émetteur, l'autre le récepteur, le graphe correspondant est orienté. Ce choix du modèle de réseau a un impact crucial sur les résultats tant théoriques qu'expérimentaux obtenus. La table 1 ci-dessous résume l'impact du type de graphe.

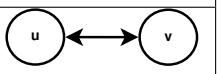
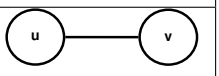
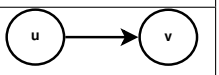
Structure	Canal	Schéma
Bi-orienté	Lien	
Non orienté	Arête	
Orienté	Arc	

TABLE 1 – Schéma de chaque modèle de réseau.

Dans la suite, nous identifions implicitement le réseau, c'est à dire le dispositif physique, avec son modèle, le graphe, en considérant que le graphe traduit la topologie sous-jacente du réseau. De plus, nous énonçons fréquemment une propriété sans préciser le modèle de réseau utilisé. Cela signifie simplement que cette dernière propriété reste valide indépendamment de la structure de graphe utilisé pour modéliser le réseau. Les terminaux d'un réseau peuvent communiquer entre eux s'il existe (au moins) un chemin depuis le nœud source vers chaque terminal du réseau. Cela signifie qu'il est possible d'acheminer un message depuis la source vers n'importe quel terminal par une suite de transmission directe entre les nœuds du réseau. Pour simplifier, nous supposons en fait que le réseau est connecté du point de vue de la source, au sens précédemment défini de la théorie des graphes. Nous supposons en outre que chaque nœud intermédiaire du réseau permet d'acheminer un message depuis la source vers au moins un terminal (sans perte de généralité puisque supprimer les nœuds ne satisfaisant pas cette dernière propriété est sans impact sur les possibilités d'acheminement de l'information au sein du réseau et peut être effectué par des méthodes efficaces). Nous allons à présent nous intéresser à l'information en elle-même, puisqu'elle constitue le cœur de notre sujet.

Modèle de flot

Étant donné un réseau comme défini précédemment, nous avons besoin d'un modèle simple et générique afin de décrire le phénomène de propagation de l'information à travers un canal de ce réseau. Puisque nous considérons explicitement la topologie du réseau, il est souhaitable que ce modèle permette de déduire certaines propriétés de l'état global du réseau à partir de la connaissance de son état local en chaque nœud. Un *modèle de flot* représente l'état de chaque canal du réseau par un nombre réel positif appelé *flot*. Ce

nombre correspond à la quantité d'informations transitant par ce canal par unité de temps (mesuré en bit par seconde, par exemple). Dans ce modèle, il est naturel de caractériser chaque canal par le débit maximum d'information pouvant transiter par lui. Cette dernière grandeur est appelée la *capacité* du canal. Définir formellement et calculer la capacité d'un canal est l'un des problèmes centraux de la *théorie de l'information* dont l'origine remonte aux travaux de Shannon [5]. Le lecteur pourra consulter l'ouvrage de Cover et Thomas [6] pour plus de détails sur le calcul de la capacité de Shannon d'un canal. Pour notre propos, il est suffisant de supposer que la capacité de chaque canal est une donnée connue, en fait un nombre rationnel positif (calculable).

Ce modèle de flot permet de formaliser une large classe de problèmes d'optimisation d'un réseau de télécommunication [7]. Il est en outre possible de concevoir des algorithmes efficaces afin de résoudre certains de ces problèmes. Il devient alors possible de répondre à des questions du type "Peut-on acheminer telle quantité d'information depuis cette source à destination de ce terminal ?", ou bien "Quelle quantité d'information devons-nous faire transiter par chaque canal afin de satisfaire une certaine demande ?", ou encore "Comment devons-nous modifier la capacité de chaque canal afin de pouvoir satisfaire telle requête ?". Aussi intéressant soit-il, ce modèle souffre d'au moins trois limitations :

Premièrement, il ne permet de réaliser qu'une *analyse quantitative* du fonctionnement du réseau, sans plus de précisions quant à l'aspect qualitatif. Ainsi, il est seulement possible de décrire *quelle quantité* d'information transite via le réseau, mais rien ne peut être dit au sujet de *ce qui* circule dans le réseau (concernant les données ou la forme des messages).

Deuxièmement, nous faisons implicitement l'hypothèse que la quantité d'information transitant par un canal est une *grandeur continue*, nonobstant la granularité intrinsèque des données. Il serait sans doute plus réaliste de considérer que le flot sur un canal est un multiple entier d'une certaine unité de débit élémentaire.

Troisièmement, le modèle de flot considéré dans cette thèse est *statique*, ou au mieux *stationnaire*. Il n'est donc pas possible de prendre directement en compte la dynamique du réseau. Il est néanmoins possible d'adapter ces modèles à un cadre dynamique par l'incorporation d'un modèle dynamique discret, au prix d'une complexité de résolution accrue. Indépendamment de ces limitations, l'approche présentée ici permet de définir for-

mellement, voire de calculer, certaines quantités utiles à la prise de décisions, comme la quantité maximum d'information qu'il est possible de transmettre via le réseau, depuis une source vers un ensemble de terminaux. En dépit de son apparente simplicité, ce modèle permet de formuler certains problèmes dont la complexité dépasse nos capacités de traitement actuelles. En pratique, ce modèle de flot fournit une intuition précieuse quant au comportement d'un réseau de télécommunications.

Nous allons maintenant détailler le modèle de capacité d'un canal en fonction du type de graphe considéré. Nous étudions un réseau avec deux nœuds u et v qui peuvent directement communiquer via un canal. Lorsque le graphe est bi-orienté, chaque arc du lien entre u et v dispose de sa capacité propre. Nous supposons en outre que la capacité de l'arc (u, v) vaut exactement celle de l'arc (v, u). Si le graphe est non orienté, chacun des deux sens de communication, de u vers v ou de v vers u, dispose d'une capacité ajustable de sorte que la somme de ces deux capacités variables égale la capacité fixe du canal. Enfin, pour un graphe orienté, chaque arc dispose d'une capacité qui lui est propre. En notant c la capacité du canal, la table 2 résume les propriétés de ce canal suivant le type de graphe.

Structure	Canal	Capacité
Bi-orienté	Lien	$c_{(u,v)} = c$ et $c_{(v,u)} = c$
Non orienté	Arête	$c_{(u,v)} + c_{(v,u)} = c$
Orienté	Arc	$c_{(u,v)} = c$

TABLE 2 – Modèle de la capacité du canal en fonction du type de graphe.

On considère à présent un exemple classique de réseau, représenté sur la figure 1, appelé *réseau papillon* / *"butterfly network"* dans la littérature (en raison de sa forme). Ce réseau est composé d'un nœud source s, de quatre nœuds intermédiaires n_1, n_2, n_3, n_4 et de deux terminaux r_1, r_2. Nous choisissons de le modéliser par un graphe orienté de sorte qu'à chaque canal est associé un arc dont la capacité est égale à 1 message par unité de temps. Cela signifie que l'envoi du message a depuis la source vers le nœud intermédiaire n_1 entraînerait l'occupation de la totalité du canal (s, n_1) pendant une unité de temps. Supposons que chacun des deux terminaux r_1 et r_2 souhaite obtenir les deux messages a et b détenus par le nœud source s. L'objectif est de communiquer *simultanément* les deux messages à chacun des deux terminaux par une suite de transmissions qui respecte

la topologie du réseau. Il est intéressant de remarquer que, si le problème consistait à acheminer les deux messages à un seul terminal, r_1 ou r_2, il suffirait de transmettre les deux messages suivant deux chemins arc-disjoints.

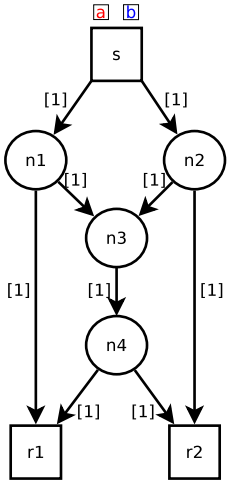


FIGURE 1 – Le réseau papillon orienté avec une capacité unitaire sur chaque arc.

Bien qu'ils lui empruntent son champ lexical, les modèles présentés jusqu'ici ne sont nullement spécifiques au domaine des télécommunications, au sens où le concept de réseau avec des capacités sur les liens est couramment utilisé pour modéliser des systèmes de distribution d'eau, de gaz, ou d'énergie, voire des réseaux biologiques. Cela nous amène à nous interroger quant aux spécificités d'un réseau de télécommunications ou, dit autrement, aux propriétés intrinsèques d'un flot d'information qui le distingue d'un flux de matière. Une des caractéristiques fondamentales de la donnée réside dans la possibilité de la manipuler, de la transformer ou de la dupliquer. Si ces propriétés ne semblent pas particulièrement attractives lorsque le problème consiste à acheminer de l'information depuis une source vers un terminal, sauf pour gérer les erreurs de transmission, nous allons voir que la possibilité d'effectuer des opérations élémentaires sur la donnée révèle tout son intérêt dans le cadre de la transmission d'un même contenu depuis une source vers un groupe de terminaux.

Opérations sur les données

Unicast

L'opération la plus simple et la plus classique qu'un nœud d'un réseau de télécommunications puisse effectuer est le transfert de données, ce qui donne naissance au principe de transmission appelé *unicast* dans la littérature des télécommunications. Le nœud reçoit un message en provenance d'un autre nœud sur un de ses canaux d'entrée et le transfère via un de ses canaux de sortie à destination d'un troisième nœud, suivant le schéma représenté sur la figure 2.

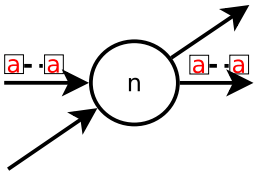


FIGURE 2 – Schéma de principe d'une transmission unicast. Le nœud intermédiaire n reçoit un flot de données comportant le message a sur un canal d'entrée et transmet le message en transférant le flot de données sur un canal de sortie.

On observe que l'ensemble des canaux utilisés pour acheminer le message depuis une source vers un terminal induit un chemin simple dans le graphe associé au réseau. L'utilisation de transmissions unicast dans le réseau papillon orienté peut donner naissance à la situation dépeinte sur la figure 3 où chaque terminal obtient le message a.

Multicast

Une autre opération élémentaire consiste à dupliquer la donnée afin d'obtenir une copie conforme du message d'origine. Il s'agit exactement du principe sur lequel repose la transmission *multicast* : au lieu de simplement transmettre la donnée reçue sur un canal d'entrée, un nœud intermédiaire du réseau peut réaliser un certain nombre de copies du message avant de transmettre chaque copie sur un de ses canaux de sortie, comme représenté sur la figure 4. Cette technique astucieuse permet d'économiser de la bande passante (la capacité consommée), puisqu'il n'est plus nécessaire d'acheminer la même information plusieurs fois afin d'atteindre tous les terminaux, comme illustré par la figure 5. De par ce principe de parcimonie, un message transitant au plus une fois par chaque canal, l'en-

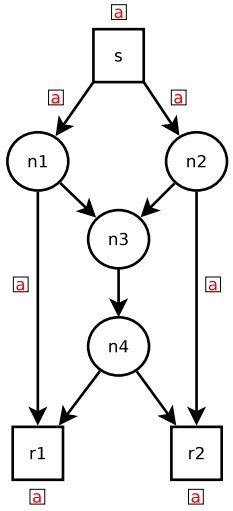


FIGURE 3 – Utilisation de la technique unicast dans le réseau papillon orienté.

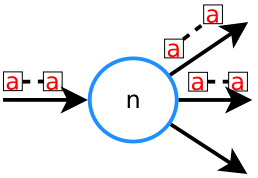


FIGURE 4 – Schéma de principe d'une transmission multicast. Le nœud intermédiaire n reçoit un flot de données comportant le message a sur un canal d'entrée, puis duplique ce message afin de transférer simultanément deux flots de données comportant le même message sur ses canaux de sortie.

semble des canaux utilisés pour acheminer un message donné depuis un nœud source vers un groupe de terminaux induit un arbre dans le graphe associé au réseau.

Cet arbre est enraciné au niveau du nœud source, couvre tous les terminaux et peut utiliser un sous-ensemble de nœuds intermédiaire afin de transmettre le message. Par la suite, un tel sous-graphe sera appelé un *arbre de Steiner* par rapport à la source et aux terminaux. Permettre à tous les nœuds du réseau papillon orienté de réaliser des opérations de copie des messages obtenus en entrée peut mener à la configuration dépeinte sur la figure 6. On voit sur le schéma de la figure 6 que, bien que les nœuds n_1, n_2 et n_4 réalisent des opérations de copie, la configuration présentée ne permet pas d'acheminer les deux messages a et b à chacun des deux terminaux, puisque le terminal r_2 n'obtient que le message b. Pour satisfaire les deux requêtes, il est nécessaire de transmettre *simultanément* les deux messages via le canal (n_3, n_4), ce qui est impossible en l'état, puisque l'envoi d'un

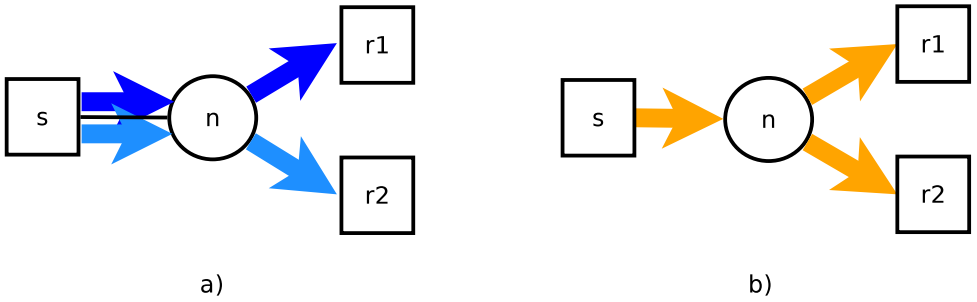


FIGURE 5 – Comparaison entre a) la technique unicast et b) la technique multicast, pour un petit réseau. En unicast, le nœud intermédiaire n peut uniquement transmettre la donnée transférée par le nœud source, de sorte que chaque message doit être envoyé deux fois via le canal (s, n) (une fois pour chaque terminal). Si l'on autorise le nœud n a dupliquer le message, grâce à la technique multicast, il suffit d'expédier une seule fois le message à travers le canal (s, n), ce qui implique que la consommation de bande passante de ce dernier canal est divisée par deux.

seul message consomme la totalité de la capacité du canal. Cela signifie que l'utilisation de la technique du multicast seule ne permet pas de résoudre notre problème de communication. Bien que cela ne constitue pas le sujet du présent document, nous souhaitons mentionner que l'utilisation de la technique multicast requiert l'emploi de protocoles de communication spécifiques. Le lecteur pourra consulter le livre de Williamson [8], l'étude d'Obraczka [9], le tutoriel par Sahasrabuddhe et Mukherjee [10], ou encore l'étude de Hosseini et al. [11] pour plus d'informations sur le multicast. Notre approche se concentre sur les problèmes d'optimisation liés à l'emploi de la technique multicast au sein d'un réseau de télécommunications. Le lecteur pourra consulter l'étude de Oliveira et Pardalos [12], qui traite précisément de ce sujet. Enfin, le multicast présenté ici est lié aux problèmes d'arbres de Steiner. Ces derniers sont étudiés dans l'ouvrage de Cheng et Du [13]. Le lecteur pourra également consulter le livre de Du et Hu [14], qui est plus spécifiquement orienté vers les problèmes de télécommunications.

Codage réseau

La notion de codage réseau / "network coding" a été formellement introduite en l'an 2000 dans le papier de Ahlswede et al. [15]. Le codage réseau repose sur l'idée d'offrir à un nœud d'un réseau de télécommunications la possibilité d'effectuer des opérations de codage. L'idée de coder un message au niveau du nœud source est assez ancienne, et

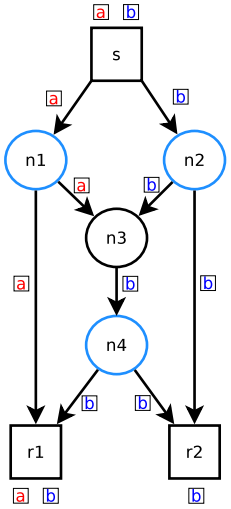


FIGURE 6 – Utilisation de la technique multicast dans le réseau papillon orienté.

permet notamment de réduire l'impact des erreurs de transmission. La nouveauté provient de ce que la donnée puisse également être codée au niveau des nœuds intermédiaires du réseau. Formellement, un nœud intermédiaire reçoit un ensemble de messages via ses canaux d'entrée. Le nœud va alors combiner ces messages entre eux de sorte à créer (au moins) un nouveau message codé, qu'il va ensuite diffusé sur ses canaux de sortie, comme illustré sur la figure 7.

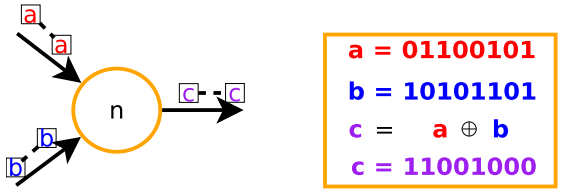


FIGURE 7 – Schéma de principe du codage réseau. Le nœud intermédiaire n obtient deux flots de données via ses canaux d'entrée, l'un des flots comportant le message a, l'autre le message b. Ce nœud va alors combiner ces deux messages afin de produire un nouveau message codé c, qu'il va par la suite diffuser sur son canal de sortie, en transmettant un nouveau flot de données. Dans cet exemple, les deux messages a et b sont représentés par un octet, de sorte que le message codé c est obtenu par application d'une opération de ou exclusif / xor sur chaque composante des messages d'entrée, suivant le tableau ci-dessous :

$\oplus$	0	1
0	0	1
1	1	0

Sauf mention explicite du contraire, nous supposerons toujours que les techniques de

codage sont déployées au sein d'un réseau où il est déjà possible d'utiliser la technique du multicast, de sorte que chaque nœud du réseau peut appliquer des opérations de duplication et de codage sur ses données d'entrée.

Nous considérons à présent la configuration dépeinte par la figure 8 dans le réseau papillon orienté. Le nœud intermédiaire n_3 effectue une opération de codage tandis que les

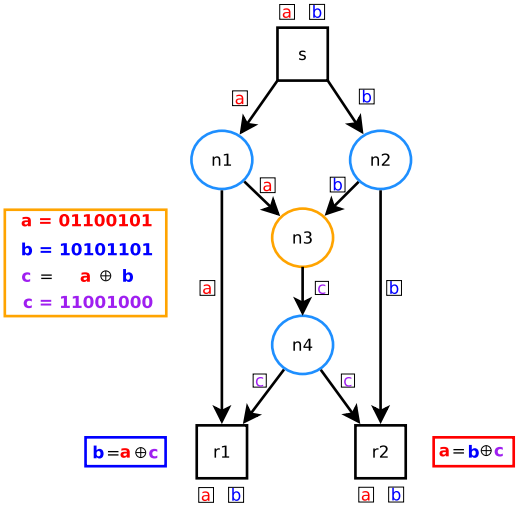


FIGURE 8 – Combinaison des techniques de multicast et de codage dans le réseau papillon orienté.

nœuds n_1, n_2 et n_4 dupliquent leur entrée sur leurs sorties (multicast). Sous réserve que chaque terminal soit capable de *décoder* le message codé c reçu à partir de l'autre message obtenu, chaque terminal va bien retrouver les deux messages a et b. En ce qui concerne le décodage, le terminal r_1 reçoit le message codé c et le message original a. Dans notre exemple, il est possible de reconstruire le message original b en effectuant à nouveau une opération de ou exclusif / xor sur chaque composante des deux messages sus-mentionnés. De même, à partir du message codé c et du message original b, le terminal r_2 peut également retrouver le message original a. Cela signifie que la combinaison des techniques multicast et codage réseau permet bien de résoudre notre problème de départ, là où le multicast seul ne pouvait fournir de solution.

Il faut cependant prendre garde à ce que l'utilisation de techniques de codage implique la conception d'un *schéma de codage*, c'est-à-dire qu'il est nécessaire de préciser pour chaque nœud du réseau les opérations de codage qu'il doit effectuer, ce qui inclut le type de code

a utiliser, afin que la suite des opérations de codage réalisées au niveau de l'ensemble du réseau permette à chaque terminal de décoder l'information. La figure 8 dépeint un schéma de codage dans le réseau papillon orienté. Cette étape de construction d'un schéma de codage s'ajoute à celle de la conception du plan d'acheminement des données à travers le réseau, qui est présente quelles que soient les techniques employées. Une présentation détaillée des techniques de codage proposées dans la littérature du codage réseau dépasse le cadre du présent document. Nous nous bornerons à renvoyer le lecteur vers le papier de Ahlswede et al. [15] ou vers l'ouvrage de Yeung [16] pour une étude du lien entre le codage réseau et la théorie de l'information. Le lecteur pourra également consulter le livre de Fragouli et Soljanin [17] et celui de Médard et Sprintson [18] pour une introduction plus complète au domaine du codage réseau. Enfin, le lecteur intéressé par le codage réseau peut aussi lire l'étude de Matsuda et al. [19] ou bien celle de Bassoli et al. [20].

Contenu de la thèse

Organisation

Le contenu du présent manuscrit est réparti à travers cinq chapitres principaux, numérotés de deux à six, le premier chapitre correspondant à l'introduction générale. Chaque chapitre traite d'un sujet propre, de sorte que les chapitres sont relativement indépendants entre eux. Cela n'exclut cependant pas l'existence de liens entre ces chapitres, représentés sur la figure 9. Nous allons à présent détailler le contenu de chaque chapitre.

Flots d'information

Le second chapitre peut être considéré comme un état de l'art des modèles de flots d'informations étudiés tout au long du manuscrit. Dans ce chapitre, nous nous concentrons sur le problème consistant à maximiser la quantité d'information qu'il est possible de transmettre depuis une source vers un ensemble de terminaux. Nous commençons par présenter le problème classique du *flot maximum* qui est au cœur de l'optimisation dans les réseaux [7]. Nous montrons ensuite comment ce problème peut être généralisé au cas d'un réseau multicast où une source délivre un même contenu à un ensemble de termi-

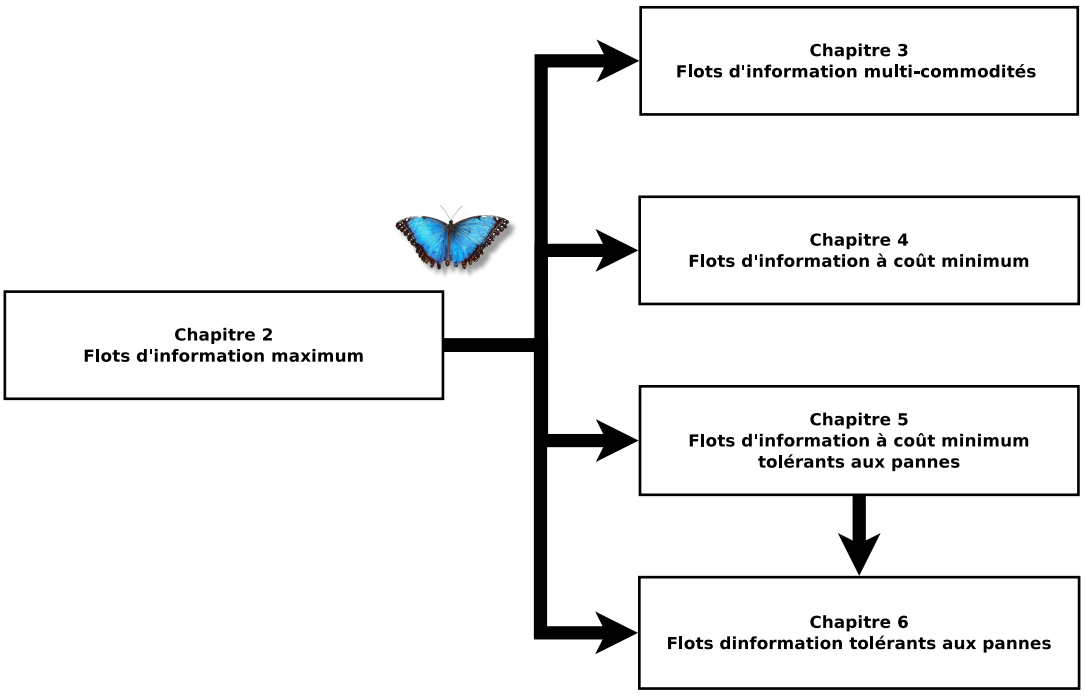


FIGURE 9 – Dépendances entre les chapitres. (Une flèche du Chapitre A vers le Chapitre B est une incitation à lire A avant B.)

naux. Pour cela, nous introduisons le concept de *flot de Steiner* qui modélise un processus d'acheminement des données via des *arbres de Steiner*. Un exemple de flot de Steiner est donné sur la figure 10. Nous montrons comment résoudre le problème du flot de Steiner maximum via un algorithme de génération de colonnes. Nous définissons ensuite un *flot codé* à partir de la littérature sur le codage dans les réseaux multicast. Nous expliquons alors comment le problème de flot codé maximum peut être résolu en temps polynomial via la programmation linéaire ou des méthodes d'optimisation dans les réseaux. Enfin, nous introduisons la notion de *gain de codage* telle qu'elle a été définie dans la littérature sur le codage réseau, afin d'évaluer l'impact de ces techniques de codage sur le débit attendu dans un réseau multicast. Nous présentons également les résultats théoriques et expérimentaux associés à ce gain de codage.

Flots d'information multi-commodités

Dans le troisième chapitre, nous présentons une extension des modèles de flots d'information présentés lors du précédent chapitre, afin de traiter du cas *multi-commodités*.

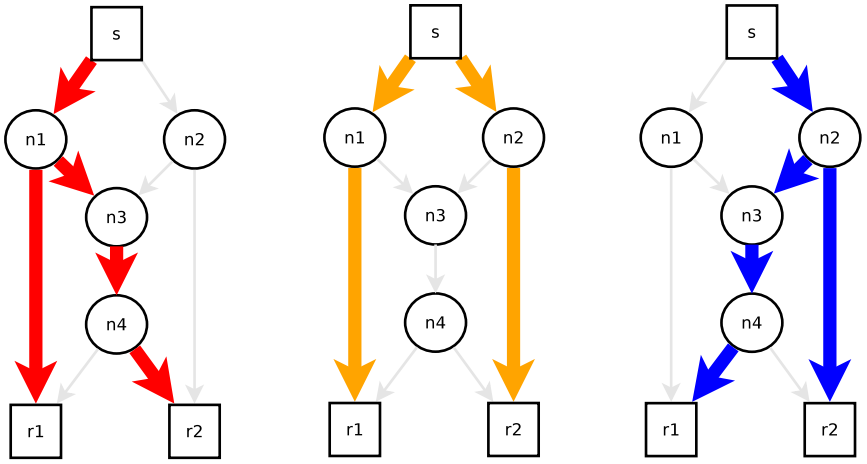


FIGURE 10 – Un exemple de flot de Steiner dans le réseau papillon orienté avec une capacité unitaire sur chaque arc. Chacun des trois arbres de Steiner permet d'acheminer $\frac{1}{2}$ unités de flot, pour un débit maximum de valeur $\frac{3}{2}$.

Une *commodité*, ou *session*, est la donnée d'une source et d'un ensemble de terminaux qui souhaitent recevoir des données en provenance de ce nœud source. Un exemple d'instance est représenté sur la figure 11. Nous supposons qu'un opérateur unique décide de l'acheminement de l'information depuis les sources vers les terminaux et de la répartition de la bande passante au niveau de chaque lien du réseau. Bien que chaque commodité soit disposée à coopérer avec les autres, nous supposons en revanche que ces dernières ne souhaitent pas voir leurs données codées avec celles des autres commodités, ce par exemple pour des raisons de confidentialité de l'information. Un exemple de schéma de codage respectant cette dernière contrainte est donné sur la figure 12.

Nous considérons alors la somme pondérée ou la proportion minimum de la demande de chaque commodité comme critères à maximiser, et nous étudions les problèmes de flot d'information associés. Nous étudions la complexité de chaque problème, et montrons comment adapter les idées présentées au précédent chapitre pour concevoir des algorithmes adaptés à ce nouveau cadre. Nous donnons également des résultats d'approximation concernant les flots de Steiner, grâce à l'approche proposée par Garg et Könemann [21], et étendue par Fleischer [22]. Nous généralisons ensuite le concept de gain de codage, tel que présenté dans le précédent chapitre, à chaque critère, puis nous dérivons des résultats théoriques et expérimentaux pour chacun des gains.

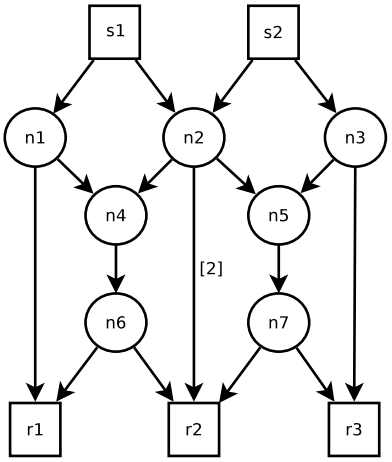


FIGURE 11 – Un réseau obtenu en fusionnant deux copies du réseau papillon orienté. La source s_1 cherche à transmettre ses données aux terminaux r_1 et r_2 tandis que la source s_2 tente de faire de même avec les terminaux r_2 et r_3. Chaque arc du réseau est muni d'une capacité unitaire sauf l'arc (n_2, r_2) dont la capacité est de 2. La transmission d'un message via un lien consomme une unité de capacité de ce lien.

Flots d'information à coût minimum

Le quatrième chapitre est motivé par le phénomène de *congestion* dans les réseaux de télécommunications. Nous adaptons nos modèles de flot d'information au problème de la recherche d'un flot de coût minimum qui satisfasse une demande donnée, dans un réseau où le délai de propagation de l'information à travers un lien est modélisé par une fonction de coût non linéaire et convexe, du type Kleinrock [23], du flot qui transite par ce lien. Un exemple d'une telle fonction de coût est donné sur la figure 13.

Nous étudions la complexité de chaque problème de minimisation du coût d'un flot d'information puis nous proposons un algorithme, basé sur la méthode du gradient et l'optimisation convexe, afin de calculer efficacement un flot d'information qui limite autant que possible l'impact du phénomène de congestion. Nous montrons explicitement comment la résolution d'un problème de flot d'information à coût minimum avec une fonction de coût *non linéaire mais convexe* peut-être ramené à la résolution d'une suite de problèmes de flot d'information à coût minimum avec une fonction de coût *linéaire*. Nous étendons ensuite la notion de gain de codage au présent contexte et nous donnons des résultats théoriques et expérimentaux sur les valeurs possibles de ce gain.

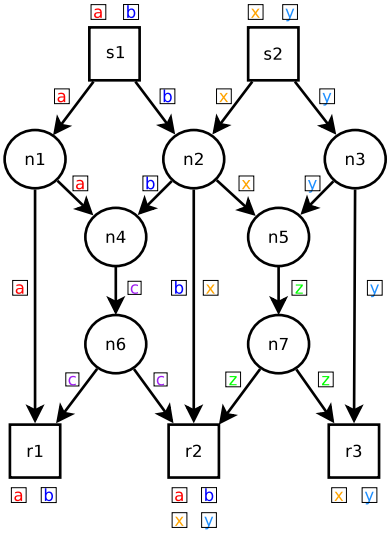


FIGURE 12 – Un schéma de codage associé à un multi-flot utilisant le codage dans le réseau précédent. Le message c est obtenu par codage des messages a et b tandis que le message z est le résultat du codage des messages x et y. Chaque terminal reçoit 2 unités de flot par commodité à laquelle il appartient.

Flots d'information à coût minimum tolérant aux pannes

Le cinquième chapitre traite de la tolérance aux pannes dans les réseaux de télécommunications. Nous étudions le problème de calcul d'un flot d'information à coût minimum, qui satisfasse une demande donnée, dans un réseau soumis à la possibilité qu'un seul lien tombe en panne. Nous supposons naturellement qu'il n'est plus possible d'acheminer de l'information à travers un lien lorsque celui-ci tombe en panne. Dans le cas d'un flot de Steiner, l'impact d'une panne d'arc sur un arbre est décrit sur la figure 14. Afin de simplifier notre étude, nous nous focalisons sur le cas où le coût de transit sur chaque lien est une fonction linéaire du flot qui le traverse. Nous cherchons alors un plan d'acheminement de l'information statique qui garantisse que la demande soit satisfaite quel que soit le lien du réseau qui tombe en panne. Nous étudions la complexité de chaque problème de minimisation du coût d'un flot d'information tolérant aux pannes et proposons un algorithme, basé sur la programmation linéaire, pour le résoudre en pratique. Nous introduisons enfin un gain de codage associé au présent contexte puis nous donnons des résultats théoriques et expérimentaux sur les valeurs possibles de ce gain.

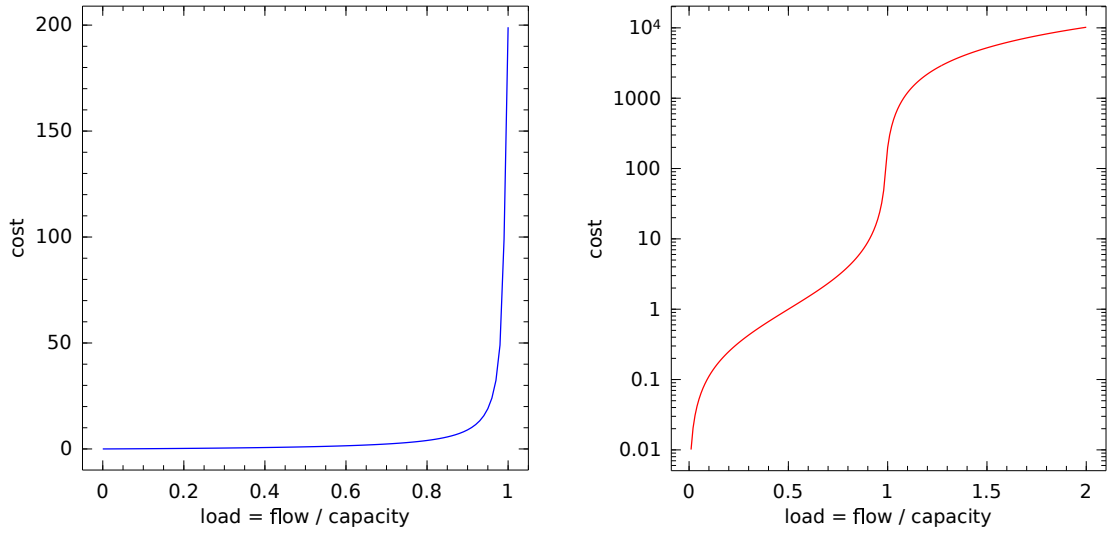


FIGURE 13 – Courbe d'une fonction de coût de type Kleinrock en fonction de la charge du lien (le ratio flot sur capacité du lien). Le coût d'acheminement de l'information via le lien devient prohibitif lorsque la charge dépasse 90%.

Flots d'information tolérant aux pannes

Dans le sixième chapitre, nous poursuivons l'étude commencé au chapitre précédent sur les problèmes de conception de flot d'information tolérant aux pannes. Dans ce chapitre, nous nous intéressons à la maximisation du débit résiduel expérimenté par chaque terminal dans le pire cas de panne d'un lien du réseau. Nous étudions à nouveau la complexité de chaque problème de maximisation du débit d'un flot d'information tolérant aux pannes et nous fournissons un algorithme, toujours basé sur la programmation linéaire, pour le résoudre efficacement. Nous mettons également l'accent sur les liens très fort qui unissent les problèmes présentés dans ce chapitre et ceux étudiés dans le chapitre précédent. En particulier, nous introduisons à nouveau un gain de codage et nous montrons que ce dernier est directement lié au gain de codage du chapitre précédent. Nous en déduisons des résultats théoriques sur les valeurs possibles de ce nouveau gain, avant de donner d'autres résultats expérimentaux. Si le chapitre précédent peut être vu comme une tentative d'évaluer le coût de prise en compte des pannes, celui-ci a pour objectif d'estimer la résistance d'un réseau face à la possibilité qu'un lien tombe en panne.

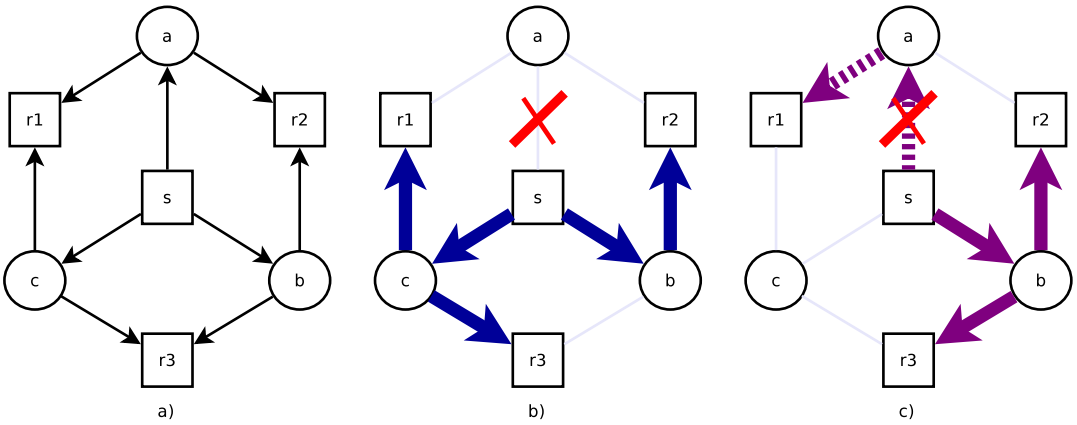


FIGURE 14 – a) Un réseau orienté avec une source et trois terminaux. b) Un arbre de Steiner qui n'utilise pas le lien (s, a) n'est pas affecté par la panne de ce lien. c) Un arbre de Steiner qui utilise le lien (s, a) ne peut plus acheminer de flot jusqu'au terminal r_1, comme suggéré par la ligne en pointillé, mais peut toujours délivrer de l'information aux terminaux r_2 et r_3.

Conclusion

Résumé

Dans cette thèse, nous étudions des modèles de flot d'information dans les réseaux de télécommunications par le prisme de l'optimisation mathématique. La possibilité de réaliser des opérations comme la copie ou le codage des données, combiné avec l'augmentation de la puissance de calcul disponible au niveau des nœuds du réseau, permet d'envisager l'utilisation de techniques d'acheminement de l'information prometteuses, comme le multicast ou le codage réseau, afin d'aider l'opérateur à faire face aux tendances qui émergent dans l'usage qui est fait de son réseau.

À cette fin, il est nécessaire de développer de nouveaux outils afin de faciliter le processus de prise de décisions du praticien quant à la conception et au choix des caractéristiques d'un réseau. L'objectif de cette thèse est donc de proposer de tels outils d'optimisation des flots d'information dans les réseaux. En focalisant notre attention sur les modèles et sur les algorithmes, nous laissons tout loisir à la communauté des praticiens des télécommunications d'évaluer les bénéfices du déploiement d'une technique donnée au sein d'un réseau particulier. Nous espérons que les outils présentés ici permettront de faciliter la résolution de certains problèmes qui se posent naturellement dans le cadre des flots d'information.

En pratique, l'utilisation de la technique du multicast est limitée par des restrictions sur le nombre d'arbres qu'il est possible de gérer lorsqu'un opérateur souhaite acheminer de l'information dans un réseau. Pour cette raison, nous focalisons notre attention sur des méthodes itératives, comme la génération de colonnes ou la descente de gradient, qui peuvent renvoyer des solutions impliquant peu d'arbres en un temps raisonnable, et nous laissons de côté d'autres types de méthodes. En ce qui concerne l'utilisation de techniques de codage, nous gardons constamment à l'esprit qu'il est dans la pratique nécessaire de concevoir un schéma de codage afin de compléter le plan d'acheminement de l'information au sein du réseau.

Pistes de recherche

Nous allons à présent suggérer certaines pistes de recherche qui nous semblent dignes d'investigation, mais que nous avons dû laisser en jachère par manque de temps.

En ce qui concerne les gains de codage, de nombreuses questions pertinentes demeurent en suspens. En particulier, la finesse des résultats théoriques démontrés dans la littérature et dans le présent manuscrit reste sujette à amélioration. Nous ignorons également s'il est possible de décider en temps polynomial du bénéfice de la mise en place de techniques de codage au sein d'un réseau multicast lorsque le critère est le gain de codage.

De même, et toujours au sujet de la complexité des problèmes étudiés, nous sommes incapables de déterminer celles des deux problèmes d'optimisation de flot de Steiner présentés aux chapitres cinq et six. Nous avons cependant démontrés que les sous-problèmes associés sont NP-difficiles. De même, nous ne savons rien quant à la possibilité de concevoir un algorithme approché pour chacun de ces problèmes. Cette question se pose également pour le problème de flot de Steiner présenté dans le quatrième chapitre. Enfin, la conception d'algorithmes décentralisés pour résoudre en temps polynomial les divers problèmes de flot codé décrits dans cette thèse est une problématique pertinente pour les praticiens des télécommunications.

Enfin, nous supposons tout au long du manuscrit que chaque nœud du réseau est capable de réaliser toutes les opérations nécessaires, comme la copie ou le codage des données. Dans la pratique, ces opérations nécessitent l'installation d'un dispositif spécifique au niveau

de chaque nœud du réseau (au minimum un composant logiciel), de sorte qu'il est sans doute plus réaliste de supposer que seul un sous-ensemble des nœuds pourra effectuer des opérations complexes, les autres nœuds restant cantonnés à un rôle de transmission des données. Ainsi, de nombreux problèmes intéressants apparaissent dans ce nouveau contexte.

Contents

Nomenclature

χ_C	Cost of a minimum-cost survivable coded flow
χ_S	Cost of a minimum-cost survivable Steiner flow
$\delta^+(v)$	Set of arcs with the vertex v as origin
$\delta^-(v)$	Set of arcs with the vertex v as destination
ℓ	A link (pair of arcs) of a bidirected network
λ_C	Value of a maximum concurrent coded flow
λ_S	Value of a maximum concurrent Steiner flow
ϕ_C	Value of a maximum multicommodity coded flow
ϕ_S	Value of a maximum multicommodity Steiner flow
ψ_C	Cost of a minimum-cost coded flow
ψ_S	Cost of a minimum-cost Steiner flow
ρ_C	Value of a maximum survivable coded flow
ρ_S	Value of a maximum survivable Steiner flow
φ_C	Value of a maximum coded flow
φ_S	Value of a maximum Steiner flow
A	Set of all arcs of a directed or bidirected network
a	An arc of a directed or bidirected network
B	A bidirected network

c_a	Capacity of the arc a
D	A directed network
d	Demand
E	Set of all edges of an undirected network
e	An edge of an undirected network
f_a	The amount of flow on the arc a
G	An undirected network
g_χ	Survivable cost coding gain
g_λ	Concurrent coding gain
g_ϕ	Multicommodity coding gain
g_ψ	Cost coding gain
g_ρ	Survivable coding gain
g_φ	Coding gain
h_a	The amount of coded flow on the arc a
L	Set of all links of a bidirected network
p	A simple path
R	A set of receivers vertices of the network
s	A source vertex of the network
t	A Steiner tree
V	Set of all vertices of a network
w_a	Weight of the arc a
x_p	The amount of flow on the simple path p
x_t	The amount of flow on the Steiner tree t
$\mathcal{P}$	Set of paths
$\mathcal{T}$	Set of Steiner trees

Chapter 1

Introduction

1.1 Motivations

Telecommunication networks and the Internet have emerged as one of the most fundamental changes in recent human history. By allowing almost-instantaneous, cheap, and reliable communication across the world, modern telecommunication drastically impacted the way we interact with others and our environment: instead of being limited to its immediate surrounding, anyone can presently interact with people or connected machines located an hemisphere away. Furthermore, although most of our everyday actions are still performed with the intent of immediately interacting locally with our surrounding environment, the possibility of broadcasting those actions may multiply, delay, or even transform their impact in an unpredictable way, to the point where an initially local phenomenon may become global through positive feedback, in the cybernetic meaning of this term, induced by the telecommunication network.

From a historical perspective, the first telecommunication devices, like messenger pigeons, the telegraph, or the phone, were designed for *point to point* communications: one sender wishes to transmit a message to a single receiver. This was also true in the early stages of the Internet. Nowadays, the increasing demand for new kinds of services, like video-streaming or live-teleconferencing, along with the now common situation where the same content is simultaneously requested by a huge number of users, stress the need for *point to many* data transmission protocols: one sender wishes to transmit the same data to a set of receivers. This evolution offers new opportunities to develop corresponding

services for various customers, but it also emphasizes the need for new ways of adapting a telecommunication network so it can cope with those required changes. One such way is to modify the routing techniques currently used in wired networks, which relies on packets forwarding. A promising way lies in allowing nodes of a telecommunication network to perform more operations than merely forwarding data.

We first provide a brief description of the core concept of a telecommunication network, then we describe how current forwarding mechanisms could be empowered to handle data distribution in a more efficient way. Finally, we summarize our goals, our methodology and the content of this dissertation.

1.2 Telecommunication network

1.2.1 Vocabulary of graph theory

Since a telecommunication network is often depicted through a graph, some basic notions of graph theory naturally appear as a prerequisite to our discussion regarding network optimization. An *undirected graph* $G = (V, E)$ is made of a finite set V whose elements are called *vertices*, along with a subset E of all pairs of distinct vertices of V, whose elements will be referred to as *edges*. A *directed graph*, or *digraph*, $D = (V, A)$ consists of a finite set V of vertices, along with a subset A of all couples of distinct vertices of V, whose elements are called *arcs*. Finally a *bidirected graph* $B = (V, A)$ is a digraph such that for any pair of vertices $\{u, v\}$ of B, the couple (u, v) belongs to the set A if and only if the couple (v, u) does. In the following, a *graph* will refer to either an undirected graph, or a digraph, or a bidirected graph. Notice those definitions implicitly assume a graph to be loop-free and devoid of multiple edges between any pair of vertices. We shall further assume that any graph under consideration has at least two distinct vertices and at least one edge. The *order* of a graph is its number of vertices while the *size* of a graph is its number of edges. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a *subgraph* of a graph $G = (V, E)$, denoted by $\mathcal{G} \subseteq G$, if $\mathcal{V}$ is a subset of V and $\mathcal{E}$ is a subset of the intersection between E and the set of all pairs / couples of distinct vertices of $\mathcal{V}$. A *path* in a graph G is a subgraph of G whose edges can be ordered in a sequence such that the extremity of one edge is the origin of the following one. A *cycle*

in a graph G is a path such that the extremity of the last edge is the origin of the first one in the aforementioned sequence. A graph is *acyclic* if it does not contain a cycle among its subgraphs. A path is *simple* if it is acyclic. A graph G is *connected* if there exists a path between any pair of vertices in G. A *tree* in a graph G is a connected acyclic subgraph of G. For a more extensive introduction to graph theory, the reader is referred to the book by West et al. [1] or the classical book by Bondy and Murty [2]. The reader interested by digraphs may have a look at the book by Bang-Jensen and Gutin [3]. Finally we would like to point out the book by Chartrand et al. [4].

1.2.2 Network topology

Informally, a *network* is a finite collection of objects, called *nodes*, which are interconnected so as to allow distant communication between a subset of nodes called *terminals*. A non-terminal node will be referred to as an *intermediate* one. In the following, a *message* will be any piece of data which is structured so as to allow a node to proceed with it. A message is made of two parts, a *header* which contains information for the network, and a *body* where the actual content of the message lies. We usually distinguish among all the terminals, a specific node called *source* node, which has all the available information, and one or more *receiver* nodes, which are requesting the aforementioned information from the source. We further assume that each node can receive a message from another one, then forward it to a third one. We say that two nodes *share a channel* if it is possible for at least one of those nodes to send a message to the other one without relying on any other node as intermediate for the transmission. Observe that our definition of a network is quite abstract and, as a consequence, a group of people randomly dispatched in a field, each one yelling at the others, would qualify as a telecommunication network according to our present description.

It is natural enough to model a network as a graph $G = (V, E)$, where V is the set of nodes and E is the set of links so that there is a link between a pair of nodes if and only if those two nodes share a channel. The structure of this graph is constrained by the considered network. If it is possible for any pair of nodes sharing a channel to communicate in both directions simultaneously, then the underlying graph will be called *bidirected*. If

instead it is required that one node is listening while the other one is sending information (they can play different roles at different times), the corresponding graph is *undirected*. Finally, if each node in any given pair has a fixed role, either sender or listener, the graph is *directed*. The type of graph used to model the network will have a huge impact on both theoretical and practical results one can obtain. Table 1.1 below summarizes this discussion.

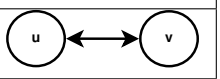
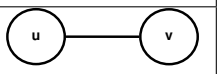
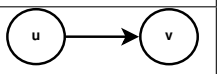
Structure	Channel	Scheme
Bidirected	Link	
Undirected	Edge	
Directed	Arc	

Table 1.1 – Scheme of each graph structure

In the following, we often implicitly identify the network, the physical device, with its model, the graph. We usually rephrase this by saying that the graph captures the underlying *topology* of the network. Furthermore, we shall often make a statement regarding a network without any explicit mention of its underlying graph structure. The reader should then understand that this particular statement holds regardless of the peculiar graph structure. A network is *connected* with respect to a given set of terminals if and only if, for any receiver, it is possible to find a path starting at the source and ending at this receiver. Hence, a network is connected if and only if it is possible to convey information from the source to any receiver through a finite sequence of messages passing between nodes. For the sake of simplicity, we shall further assume the network to be connected in the classical graph-theoretical sense and that any intermediate node in the network can be used by the source to convey data to at least one receiver. We shall now focus our attention on the information itself, which lies at the core of telecommunication.

1.2.3 Flow model

Given a network consisting of at least two nodes sharing a channel, we are looking for a simple and generic way to describe the information propagation phenomenon taking place

along a channel. Since we are explicitly dealing with the network topology, we are looking for a model which allows to deduce global properties from local ones: given the state of each channel of the network, we should be able to fully characterize the state of the whole network itself. Roughly speaking, a *flow model* summarizes the state of any channel by a single positive real number called its *flow* or *rate*, corresponding to the amount of data going through this channel every unit of time (expressed in bits per second for example). Hence it becomes possible to describe the channel by a single quantity, called *capacity*, which is the maximum rate this channel can sustain. Properly defining the capacity of a channel is one of the main problems of the field called *information theory*, taking its roots in the pioneering work of Shannon [5]. We refer the reader interested in digging into the information theoretical definition of a channel capacity to the book by Cover and Thomas [6]. For our purpose it will be sufficient to assume we are given, for each channel of the network, a positive capacity, which we further assume to be a rational number a computer can easily deal with. The proposed flow model allows one to formalize, and hopefully to solve efficiently, a large class of network optimization problems [7]. It becomes possible to answer questions like: "Is it possible to route this amount of flow from this source node to this receiver node?", "What amount of flow should we route through each channel so as to satisfy some given demand?", or "How should we modify this channel capacity so as to meet a given request?". This simple model has at least three drawbacks: Firstly, it only allows for a quantitative analysis of the behavior of a network, without more consideration for the content itself. Hence, it is only possible to describe *how much* information is going through the network, but nothing can be said about *which* information is actually conveyed and the message structure cannot be encompassed by this approach. Secondly, we implicitly assume the amount of information flowing through a channel is a continuous quantity notwithstanding the actual granularity of data. To tackle this issue, it would be more realistic to consider an integral number of messages by unit of time. Thirdly, this flow model is either *static*, or at best *stationary*. Hence, it cannot deal with any *dynamical* aspect of the network behavior. Regardless of such limitations, this approach allows one to at least properly, read mathematically, define, then hopefully compute, some interesting quantity like, for example, the average amount of information which can be conveyed from

the source to any receiver in the network. Despite its crude aspect, we will see that this model often gives rise to problems whose computational complexity by far exceeds our ability to deal with them. Nonetheless, this simple flow model also provides useful insights on the behavior of practical networks.

We shall now describe our model of channel capacity as a function of the graph structure. Consider a network with at least two nodes u and v sharing a channel. When the graph associated to this network is bidirected, we model the channel between nodes u and v by a pair of directed arcs, one arc (u, v) from u to v, the other one in the reverse direction, (v, u) from v to u. Each arc is equipped with its own capacity but we require the two capacities on the arcs (u, v) and (v, u) to have the same value. However, when the graph is undirected, the channel is modelled by a single edge with a capacity shared by the two possible flows, one going from u to v, the other one from v to u. Finally, when the graph is directed, assuming it is only possible to use the channel to convey information from u to v, the model is given by a single arc (u, v) with a capacity value. Assuming any channel is equipped with a capacity c, Table 1.2 summarizes the capacity model of each structure.

Structure	Channel	Capacity
Bidirected	Link	$c_{(u,v)} = c \quad c_{(v,u)} = c$
Undirected	Edge	$c_{(u,v)} + c_{(v,u)} = c$
Directed	Arc	$c_{(u,v)} = c$

Table 1.2 – Capacity model of each graph structure.

Although it is very common in the network optimization literature that the directed version of a problem encompasses the undirected one, we would like to point out that this will *not* always be the case along the present document. In the following, we will precise the model hierarchy of each problem.

As an example, consider the network depicted on Figure 1.1, which we will refer to as the *directed butterfly network* (the name comes from its butterfly shape), made of one source s, four intermediate nodes n_1, n_2, n_3, n_4, and two receivers r_1 and r_2. Here, we choose to model this network by a digraph. Each arc, corresponding to one of the nine existing channels, has a unit capacity, measured in number of messages by unit of time that can go through it. Hence, sending message a emitted by at the source on any

of its two output arcs, would consume the whole capacity of this arc for one time step. Assume each receiver requests the two messages a and b, so one is looking for a way to *simultaneously* convey both messages to each receiver by performing a sequence of step by step data transfers according to the network topology. Notice that should only one receiver actually requests the two messages, it would be easy to convey both messages by using two arc-disjoint paths. Observe that, up to now, our discussion over telecommunication

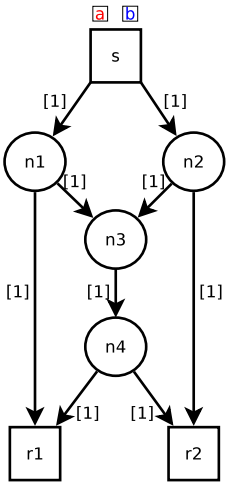


Figure 1.1 – The directed butterfly network with unit capacity on each arc.

networks only makes use of telecommunication itself as a semantic field: the very notion of a network with capacity on the channels can be and has been applied to model many problems outside the field of telecommunication, ranging from water distribution systems to biological networks. However, as already pointed out by Cover and Thomas, "*the theory of information flow in networks does not have the same simple answers as the theory of flow of water in pipes.*" [6]. This important remark leads us to search some fundamental features of an information flow. Thus, we are inquiring about the possibility offered by a flow of data with respect to another common type of flow conveying a resource like water, gas, or electricity. A fundamental feature of data is that it can be freely manipulated, transformed, stored, and even cloned. Although this does not seems to be interesting if one wishes to send data along a network from a given source to a single destination, except for transmission losses handling, it appears that the ability to perform operations on data inside the network gives rise to promising applications in the case one wishes to send the

same data from one source to a set of receivers. We shall now give more details on this latter point.

1.3 Performing operations on data

1.3.1 Unicast

The most classical operation performed by any node of a telecommunication network is to forward data. This principle of forwarding, referred to as *unicast* in the telecommunication literature, means a node simply transmits on only one of its output channels a message previously obtained from one of its input channels, as depicted in Figure 1.2. Observe that the set of edges used to convey a given message from the source to a receiver

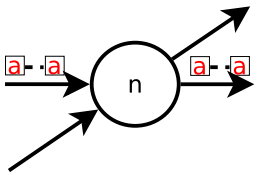


Figure 1.2 – Scheme of the unicast principle. The intermediate node n receives a stream of packets with message a from one of its input channels, then forwards this message by releasing the stream of packets on one of its output channels.

induces a simple path in the network between the source and this peculiar receiver. Applying unicast techniques in the directed butterfly network may lead to the situation depicted in Figure 1.3 where both receivers get message a.

1.3.2 Multicast

A simple operation one could perform with data is to replicate it so as to obtain a perfect copy. This is exactly the principle underlying *multicast*: instead of merely forwarding the received information, any intermediate node of the network is allowed to perform copies of the data conveyed by its input channels, before simultaneously releasing those copies on its output channels, as emphasized in Figure 1.4. This powerful technique allows one to spare the precious bandwidth, or used capacity, since it is no longer required to send the same information multiple times in order to reach each receiver, as depicted in Figure 1.5. Because of this property of sparsity, any information being transmitted only once through

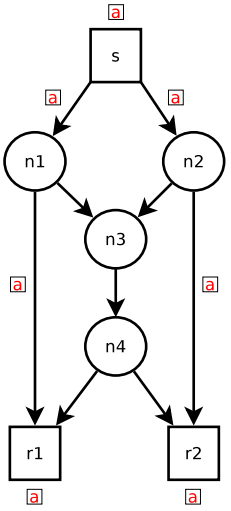


Figure 1.3 – Using the unicast technique inside the directed butterfly network.

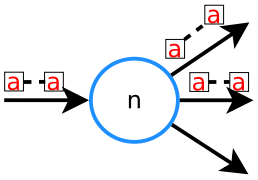


Figure 1.4 – Scheme of the multicast mechanism. The intermediate node n first receives a stream of packets with message a from its input channel, then it replicates this message so as to create several streams of packets with the same message a which are simultaneously released on its output channels.

a given edge, the set of used edges associated to the sequence of all transmissions of a particular piece of data, from the source to the set of receivers, typically induces a subgraph taking the shape of a tree, rooted at the source and spanning all receivers along with some of the intermediate nodes. In the following a *multicast tree*, or *Steiner tree*, will refer to such a construction. Back to the directed butterfly network, allowing all nodes to perform multicast operations could lead to the configuration depicted in Figure 1.6. Observe that, in Figure 1.6, although the network replicates data at intermediate nodes n_1, n_2, and n_4, the depicted configuration does not satisfy our expected requirement that both receivers should get the two messages a and b, since receiver r_2 only gets message b. To satisfy all requests, one should be able to *simultaneously* send both messages through channel (n_3, n_4) which is impossible since one message consumes the whole channel capacity. Hence, the multicast technique alone appears insufficient to solve this particular forwarding problem. Although

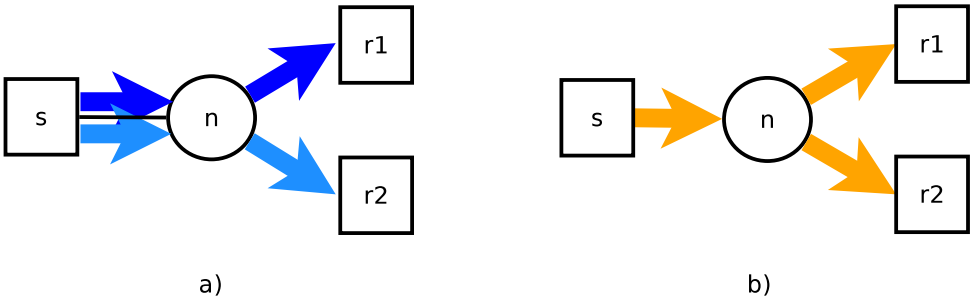


Figure 1.5 – Comparison of a) the unicast setting and b) the multicast setting, on a small network. In unicast the intermediate node n can only forward data received from the source. Hence, any message has to be sent twice on the channel (s, n), once for each receiver. By allowing node n to perform a multicast operation, replicating the received data, it becomes sufficient to send the message only once on channel (s, n), effectively halving the consumed bandwidth on this channel.

this manuscript does not delve into this direction, the multicast technique requires specific protocols so as to convey information through a network. We refer the reader to the book by Williamson [8], the survey by Obraczka [9], the tutorial by Sahasrabuddhe and Mukherjee [10], and the survey by Hosseini et al. [11] for more information regarding this topic. Instead, the present work focuses on multicast as a combinatorial problem. The interested reader may read the survey by Oliveira and Pardalos [12] which specifically deals with this latter topic. Since combinatorial multicast involves the use of Steiner trees, the classical book by Cheng and Du [13] along with the more telecommunication focused one by Du and Hu [14] may also be useful.

1.3.3 Network coding

The concept of *network coding* has been formally introduced in a breakthrough research paper by Ahlswede et al. in 2000, see [15]. The main idea behind network coding is to allow any node in the network, especially intermediate ones, to perform *coding operations*. Notice that the idea of encoding a message at the source so as to reduce errors during the transmission process is quite old. The novelty of network coding lies in the idea of allowing coding to take place *inside* the network. An intermediate node receives a set of messages from its input channels, and instead of directly releasing the information, the node creates a bunch of new messages from the obtained data thanks to a chosen coding mechanism

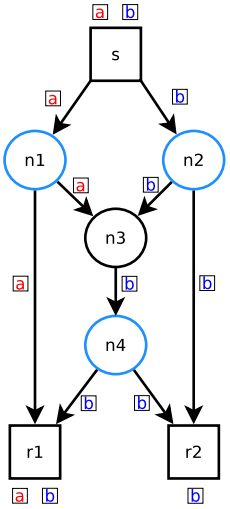


Figure 1.6 – Using the multicast technique inside the directed butterfly network.

applied to the original received messages, see Figure 1.7. Unless explicitly mentioned, we

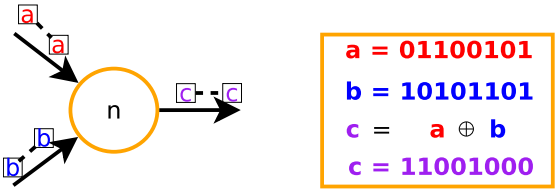


Figure 1.7 – Scheme of the network coding mechanism. The intermediate node n first receives two streams of packets with messages a and b from its input channels. It then combines those two messages so as to produce a new coded message c. This newly created message is afterward released on the output channel of node n through a stream of coded packets. In this example, assuming that the two messages a and b are actually encoded as binary strings of fixed size, the coded message c can be created by performing a xor operation component-wise, according to the following table:

$\oplus$	0	1
0	0	1
1	1	0

will always assume in the sequel that the coding process takes place in a network where the multicast setting is allowed. Hence, it is possible to freely perform both replicating and coding operations at any node inside the network. Back to our forwarding problem in the directed butterfly network, we now focus our attention on the configuration depicted on Figure 1.8. Here, the network performs a coding operation at intermediate node n_3 along with replicating data at intermediate nodes n_1, n_2, and n_4. Assuming that each

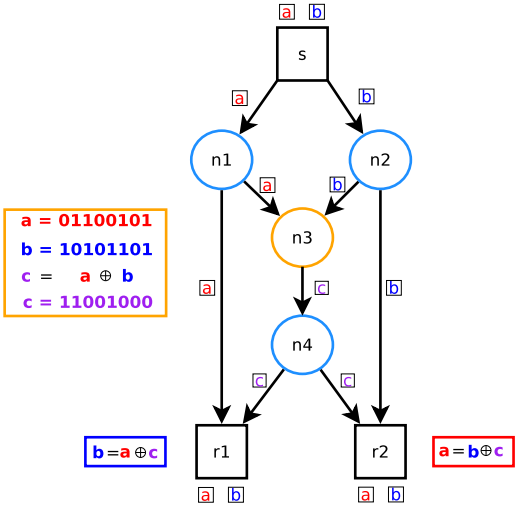


Figure 1.8 – Using a combination of multicast and network coding techniques in the directed butterfly network.

receiver is able to *decode* message c, by using the other non-coded message it gets, both receivers will obtain the two messages a and b. Given non-coded message a and coded message c, receiver r_1 combines those two messages to get message b back, while receiver r_2 computes message a from non-coded message b and coded message c. In our example, it is sufficient to perform component-wise xor operations on the two messages obtained by each receiver to get the missing one back. Hence, by simultaneously performing duplicating and coding operations in the network one can solve this forwarding problem, while using multicast alone is insufficient. Notice however that the implementation of network coding involves the design of a *coding scheme*. Not only has one to decide how data should be conveyed from the source to each receiver along the network (as in unicast or multicast) but one also has to consider how to code/decode the data at each intermediate node of the network (namely, what kind of code should be used), while ensuring that each receiver is able to properly decode the coded information delivered by its input channels. As an example, Figure 1.8 can be regarded as a depiction of a coding scheme used to convey the two messages a and b to both receivers in the directed butterfly network. Since a survey of the coding techniques which have been proposed in the network coding literature is way beyond the scope of this manuscript, we will instead provide some useful references. In addition to the seminal work of Ahlswede et al [15], the reader looking for a connection

between information theory and network coding may have a look at the book by Yeung [16]. For a good introduction to network coding, the reader can review the book by Fragouli and Soljanin [17] or the more recent one by Médard and Sprintson [18]. Finally, the reader may also be interested in the survey by Matsuda et al. [19] or the one by Bassoli et al. [20].

1.4 Content of the PhD

1.4.1 Goals

The work conducted during this PhD had two main goals. First, trying to formalize some problems naturally arising in the field of telecommunications by using operations research and mathematical optimization tools. Our objective is to develop models and algorithms which could either compute or at least approximate some quantities which we consider relevant as far as forwarding data in networks is concerned. The second goal is to evaluate, both in theory and numerically, the impact of network coding techniques in a multicast network. This means we compare, for a given network, the two settings of multicast alone versus multicast with network coding.

We specifically focus our attention on combinatorial multicast and combinatorial network coding, with much less emphasis put on some very important aspects like multicast routing protocols, network code design, or the information theoretic notion of network capacity. Hence this manuscript should be regarded as lying in the field of network optimization rather than in the one of telecommunications, although the latter plays a crucial role in the sequel by providing insights on the relevance of the proposed models. From a more practical perspective, we are interested in applications regarding live streaming services like video-conferencing or multi-users content distribution. Hence our approach deals exclusively with the setting where any receiver is requesting the same content from one unique source. We would like to point out that relaxing any of those restrictions may quickly lead to intractable problems as emphasized by Lehman and Lehman [24], Cassuto and Bruck [25], Dougherty et al. [26], Chekuri et al. [27], and Langberg and Sprintson [28].

We focus our attention on single-session routing problems where the set of receivers is fixed, leaving apart the practical dynamic version of those problems where one or more nodes may join or leave the set of current receivers. We also restrict our study to wired networks, while we would like to mention that a huge literature on wireless and mobile networks has been produced for both multicast alone and multicast with network coding. Finally, coding techniques can be deployed in a network without multicast, a setting referred to as *multiple-unicast* framework in the relevant network coding literature, although the corresponding problems are considerably harder than their multicast counterparts, see [29].

1.4.2 Methodology

Our methodology can be roughly described as follows: We first identify a relevant criterion with respect to the field of telecommunications, like, for example, the maximum amount of information one can expect to convey from the source to a set of receivers through the network, or the minimum amount of network congestion one can guarantee while satisfying a given demand. Once this choice has been made, we formalize the two corresponding forwarding problems thanks to operations research and network optimization techniques: One problem corresponding to the setting where multicast alone is considered, another problem to deal with the case where both multicast and network coding can be simultaneously used. We then design an algorithm whose purpose is either to compute an optimal solution if possible or to find a feasible solution with a provable guarantee on its quality. We finally discuss the impact of the network structure on the benefit one can expect from introducing network coding mechanisms in a multicast network.

1.4.3 Dissertation outline

The core of the present manuscript is divided into five chapters, referred to as the main chapters, and followed by a last concluding chapter. Each of the main chapters is itself made of an introduction, followed by the main body, and ended by a conclusion. Each main chapter deals with its own topic so that the five aforementioned chapters are relatively independent. Beware however since some connections do exist between them.

The scheme depicted on Figure 1.9 below provides a summary of those connections.

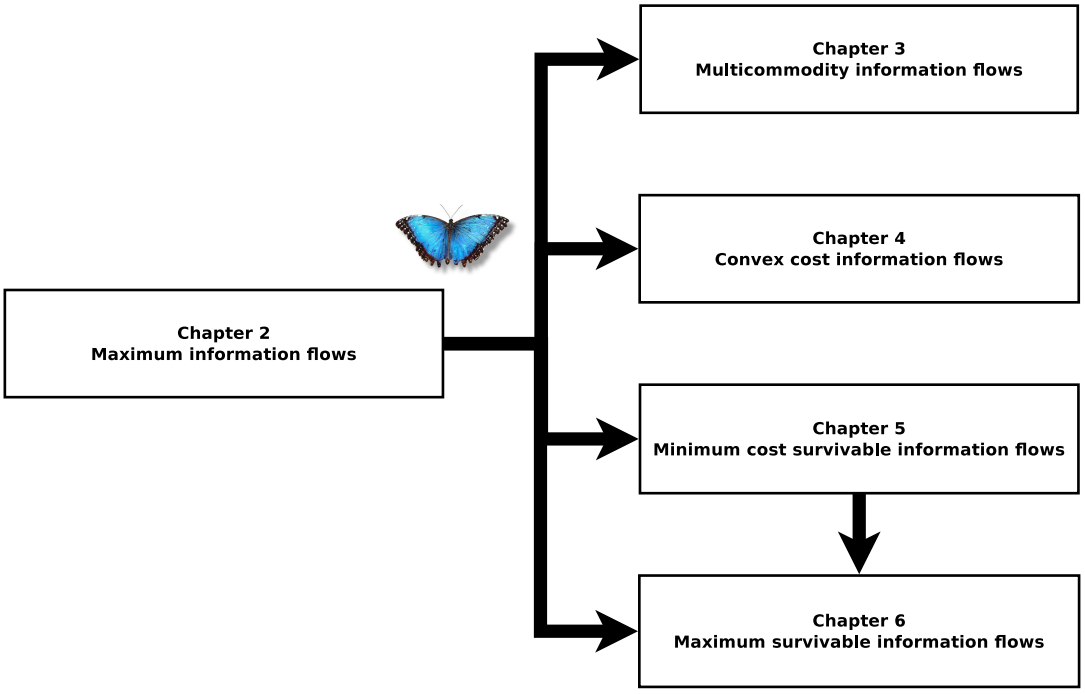


Figure 1.9 – Dependencies between chapters. (An arrow from Chapter A towards Chapter B is an incentive to read A before B.)

The second chapter can be thought of as a state-of-the-art regarding network optimization and information flows. We begin by presenting the maximum flow problem. We then formally define the two types of *information flows* which we will deal with throughout this document, namely the *Steiner flow* and the *coded flow*. The topic of interest in this chapter is how to maximize the throughput experienced by all receivers in the network.

In the third chapter we present an extension of the information flow framework to the setting where the network is simultaneously handling more than one *commodity* or *session*. The topic of interest is to study the maximum amount of flow which can be conveyed in a network where the operator wishes to ensure some form of fairness among the commodities. We study the complexity and approximability of some multicommodity information flow problems, and we extend some classical results in the network coding literature to the multicommodity setting by expanding the analysis presented in the second chapter.

The fourth chapter is motivated by *congestion* issues in telecommunication networks. We show how to extend to the information flow framework the classical problem of finding

a *minimum cost flow* in a network where each channel is associated with its own *convex cost function*. We present algorithms based on the conditional gradient to solve each minimum cost information flow problem. We provide theoretical bounds on the gain one can expect from using network coding in this setting by combining some convex analysis with classical results in the network coding literature regarding minimum-cost information flows where the cost function is linear.

The fifth chapter deals with *survivability* and *failure tolerance* issues in a telecommunication network. We study the problem of finding a *minimum cost information flow* while satisfying an explicitly given demand in a network prone to *single channel failures*. For the sake of simplicity we focus the study on the setting where the cost function of each channel is *linear*. We propose models and algorithms, based on linear programming, to deal with this issue. The content of the fifth chapter can be regarded as an extension of classical results in the network coding literature to the framework of a network prone to failures.

In the sixth chapter, we pursue the study begun in the fifth chapter by considering the problem of maximizing the *residual amount of flow* which can be routed by an information flow in the worst-case of a *single channel failure*. We again propose models and algorithms, based on linear programming, to deal with this issue. We also underline the strong connection between the problems studied in this chapter and those presented in the previous one. While the fifth chapter can be regarded as an attempt to evaluate the cost of handling failures, the purpose of the sixth chapter is to estimate the *survivability* of a network with respect to single channel failures.

Chapter 2

Maximum information flows

2.1 Introduction

2.1.1 Motivation

This chapter draws a state-of-the-art for the simple setting where one wishes to find the maximum amount of data one can expect to convey from a source node to a set of receiver nodes, thanks to a telecommunication network where multicast and coding techniques may be freely used at any node. Throughout this chapter, we attempt at providing a unified view of many well-known results in the network coding literature.

2.1.2 Content

We first present the famous *maximum flow problem* which lies at the core of network optimization [7]. This problem has an interest of its own, highlighting the impact of a model formulation, and it will also play a crucial role in the whole chapter. The second problem we shall present, referred to as the *maximum Steiner flow problem*, can be thought of as a generalization of the maximum flow problem under path formulation, with Steiner trees replacing paths as the basic components of the flow. We will provide a complete study of the maximum Steiner flow problem, with a particular emphasis on its main application as a way to compute the maximum amount of data one can convey through a multicast network. The third problem we shall consider, namely the *maximum coded flow problem*, takes its roots in the network coding setting, where one wishes to establish the maximum amount of data which can be conveyed by performing coding operations at intermediate

nodes of a multicast network. It turns out that this last problem is tightly connected to the maximum flow problem. We then introduce the notion of *coding gain* as an indicator of the benefit, in terms of maximum achievable throughput, one can expect from using coding techniques in a multicast network. We review various results from the network coding literature regarding the values one can expect the coding gain to take as a function of the network structure. The reader already familiar with the *maximum flow problem* may skip the remaining of this introduction.

2.1.3 Maximum flow and the notion of throughput

2.1.3.1 Literature review

The so-called *maximum flow problem* is arguably one of the most fundamental problems in the field of network optimization. Laying its roots in the Fifties with the pioneering works of Harris and Ross [30], Ford and Fulkerson [31], or Elias et al. [32], this problem enjoys constant attention from both practitioners and the academic community since then. Recent breakthrough papers on this particular problem include but are not limited to [33, 34, 35, 36]. We refer the reader to the classical book by Ahuja et al. [7] for a general presentation of various algorithms solving this problem. See also the paper by Schrijver [37] for a historical perspective.

2.1.3.2 Arc formulation

We are given a directed network $D = (V, A)$ with a special vertex s called the *source* and another vertex r, distinct from s, called the *receiver*. Assume there is at least one path from the source to the receiver. Also assume we are given, for each arc a of the network, a non-negative real number c_a called the *capacity* of a. In the following, we denote by $\delta_D^-(U)$, respectively $\delta_D^+(U)$, the set of arcs *entering*, respectively *leaving*, a set of vertices $U \subseteq V$.

$$\delta_D^-(U) = \{(v, u) \in A : v \in V \setminus U, u \in U\} \tag{2.1}$$

and

$$\delta_D^+(U) = \{(u, v) \in A : u \in U, v \in V \setminus U\} \tag{2.2}$$

We drop the subscript D when it is clear from the context. When the set U is a singleton, $U = \{u\}$, we use the slight abuse of notation $\delta^-(u)$ and $\delta^+(u)$.

A *flow* f is a function from the set of arcs A to the set of non-negative real numbers $\mathbb{R}_+$. The quantity f_a is called the *amount of flow on arc* a. A flow f satisfies the *capacity requirements* if, for each arc a, the amount of flow f_a on a is at most the value of the capacity c_a, namely $f_a \leq c_a$. Furthermore, a flow f meets the *conservation requirements* if, for each vertex v different from the source s and the receiver r, the amount of flow entering v equals the amount of flow leaving v:

$$\sum_{a \in \delta^-(v)} f_a = \sum_{a \in \delta^+(v)} f_a \tag{2.3}$$

Those conservation requirements prohibit both the creation and the destruction of flow at any vertex different from the source or the receiver. The *throughput*, or *value*, $|f|$ of a given flow f is the *net* amount of flow leaving the source, or equivalently, from the conservation requirements, the *net* amount of flow entering the receiver:

$$|f| = \sum_{a \in \delta^+(s)} f_a - \sum_{a \in \delta^-(s)} f_a = \sum_{a \in \delta^-(r)} f_a - \sum_{a \in \delta^+(r)} f_a \tag{2.4}$$

Let $\mathcal{F}$ be the set of all flows satisfying both the capacity and conservation requirements. The maximum flow problem is to find a flow whose throughput is maximum:

Problem	*Maximum flow*
Instance	Network $D = (V, A)$, source s, receiver r, capacity c_a for each arc a
Solution	A feasible flow $f \in \mathcal{F}$ in D
Objective	Maximize the value $\|f\|$ of flow f as defined in Equation (2.4)

We denote by φ_F the value of a maximum flow for the considered instance:

$$\varphi_F = \max_{f \in \mathcal{F}} |f| \tag{2.5}$$

It can be shown that, if the capacity c_a of each arc a is an integer, then there exists a maximum flow f such that its amount on any arc is integer, so that its throughput also has an integer value. Although this problem has been presented so far in the setting of directed networks (for the reader's convenience), it is possible to properly define it in the setting of undirected networks. Given an undirected network $G = (V, E)$ with source s,

receiver r, and capacity c_e for each edge e, consider the bidirected network $B = (V, A)$ associated to G by replacing each edge $\{u, v\}$ by the pair of arcs (u, v) and (v, u). Also replace the previous capacity requirements by $f_{(u,v)} + f_{(v,u)} \leq c_{\{u,v\}}$ for each edge $\{u, v\}$.

The definition above is referred to as the *arc formulation* of the maximum flow problem so as to distinguish it from the *path formulation* of the same problem which we shall present below. In the remaining of this thesis, we often model an optimization problem using a framework called *linear programming*. We refer the interested reader to Appendix A.1 for a brief review of this framework. The arc formulation of the maximum flow problem yields the following linear program:

$$\begin{cases} \varphi_F = \max \quad \varphi & (2.6) \\ \text{s.t.} \quad f_a \leq c_a \quad \forall a \in A & (2.7) \\ \quad \sum_{a \in \delta^+(v)} f_a - \sum_{a \in \delta^-(v)} f_a = b_v(\varphi) \quad \forall v \in V & (2.8) \\ \quad \varphi, f_a \geq 0 \quad \forall a \in A & (2.9) \end{cases}$$

with

$$b_v(\varphi) = \begin{cases} \varphi & \text{if } v = s \quad (2.10) \\ -\varphi & \text{if } v = r \quad (2.11) \\ 0 & \text{otherwise} \quad (2.12) \end{cases}$$

For each arc a, variable f_a stands for the amount of flow going through a, while variable φ models the throughput of flow x. Constraints (2.7) ensure that the flow satisfies all capacity requirements, while Constraints (2.8) enforce the conservation of flow at each intermediate vertex of the network.

2.1.3.3 Path formulation

We choose to present this formulation in the setting of directed networks so as to emphasize the connection with the previous one. Notice however that the path formulation of the maximum flow problem is transparent to any kind of network model (digraph or undirected graph). Let $D = (V, A)$ be a directed network, with a source vertex s, a receiver vertex r, and a capacity c_a on each arc a. We denote by $\mathcal{P}$ the set of all simple (loop-free) paths from the source to the receiver. Also let $\mathcal{P}_a$ be the subset of such paths going through arc a.

In this context, a *flow* x is a function from the set of paths $\mathcal{P}$ to the set of non-negative real numbers $\mathbb{R}_+$. The quantity x_p is called the *amount of flow on path* p. Given a flow x under path formulation, one can define a flow f under arc formulation by setting:

$$f_a = \sum_{p \in \mathcal{P}_a} x_p \tag{2.13}$$

Flow x is then called *a path decomposition* of flow f, while vector f will be referred to as *the arc projection* of x. A flow x satisfies the *capacity requirements* if and only if its arc projection f does, namely $f_a \leq c_a$ for each arc a. Observe that the previous conservation requirements are implicitly satisfied in this formulation (see Appendix A.2 for a formal proof). In this model, the *throughput* $|x|$ of a given flow x is the sum of the contributions of all paths:

$$|x| = \sum_{p \in \mathcal{P}} x_p \tag{2.14}$$

Notice that this definition of the throughput of flow x is compatible with the previous one, namely $|x| = |f|$, where vector f stands for the projection of flow x (see Appendix A.2 for a formal proof). We denote by $\mathcal{X}_\mathcal{P}$ the set of all flows satisfying the capacity requirements:

$$\mathcal{X}_\mathcal{P} = \left\{ x \in \mathbb{R}_+^\mathcal{P} : \sum_{p \in \mathcal{P}_a} x_p \leq c_a \, \forall a \in A \right\} \tag{2.15}$$

The maximum flow problem still consists in finding a flow whose throughput is maximum:

Problem	*Maximum flow*
Instance	Network $D = (V, A)$, source s, receiver r, capacity c_a for each arc a
Solution	A feasible flow $x \in \mathcal{X}_\mathcal{P}$ in D
Objective	Maximize the value $\|x\|$ of flow x as defined in Equation (2.14)

We denote by $\varphi_\mathcal{P}$ the value of a maximum flow for the considered instance:

$$\varphi_\mathcal{P} = \max_{x \in \mathcal{X}_\mathcal{P}} \sum_{p \in \mathcal{P}} x_p \tag{2.16}$$

The path formulation of the maximum flow problem also yields a linear program:

$$\begin{cases} \varphi_\mathcal{P} = \max \quad \displaystyle\sum_{p \in \mathcal{P}} x_p & (2.17) \\ \text{s.t.} \quad \displaystyle\sum_{p \in \mathcal{P}_a} x_p \leq c_a \quad \forall a \in A & (2.18) \\ \qquad x_p \geq 0 \quad \forall p \in \mathcal{P} & (2.19) \end{cases}$$

For each simple path p, the variable x_p stands for the amount of flow conveyed through p. Observe that, since we may have an exponential number of paths in $\mathcal{P}$, this model may have an exponential number of variables. Constraints (2.18) ensure that flow x satisfies all capacity requirements.

The arc and path formulations of the maximum flow problem are equivalent with respect to the value of a maximum flow, namely $\varphi_F = \varphi_{\mathcal{P}}$ (see Appendix A.2 for a formal proof).

2.1.3.4 Complexity and algorithms

The maximum flow problem can be solved to optimality in strongly polynomial time, with respect to the instance size (say the order n and the size m of the network), thanks to the Edmonds-Karp algorithm [38]. Interestingly, this algorithm provides a feasible solution to both formulations. See the book by Ahuja et al. [7] for a comparison between various algorithms solving this problem.

2.1.3.5 Minimum cut

The dual of the maximum flow problem is called the *minimum cut problem*. One is looking for an assignment of a non-negative value on each arc of the network so as to minimize the overall weighted sum of those values, where the weight of each arc a is its capacity c_a. Furthermore, the assignment must ensures that the length of any path from the source to the receiver is at least 1, where the length of each arc is precisely its assigned value. It turns out that there is always an optimal solution to the minimum cut problem such that the value assigned to each arc a of the network is either 0 or 1. Thus, a *cut* can be identified with a subset of arcs. It is also possible to define a *cut* between the source s and the receiver r as a subset of vertices U such that $s \in U$ and $r \in V \setminus U$. Those two definitions are compatible since, given a cut U with respect to the latter definition, the set of arcs $\delta^+(U)$ is a cut with respect to the former one.

2.1.3.6 A detailed example

Consider the small directed network depicted on Figure 2.1 (a). A throughput of 3 (measured in the same unit as the capacities) can be conveyed from the source s to the

receiver r by a maximum flow f in this network, as highlighted on Figure 2.1 (b). On Figure 2.1 (c), a path decomposition x associated to the maximum flow f is depicted. Notice that each path conveys 1 unit of flow. A minimum cut between the source s and the receiver r is provided on Figure 2.1 (d). It can be checked that any path in the network has an incurred length of value at least 1 with respect to the value on each arc. Moreover, the overall weighted sum of those values equals 3 (where the weight of each arc is its capacity). The optimality of this cut follows by weak duality.

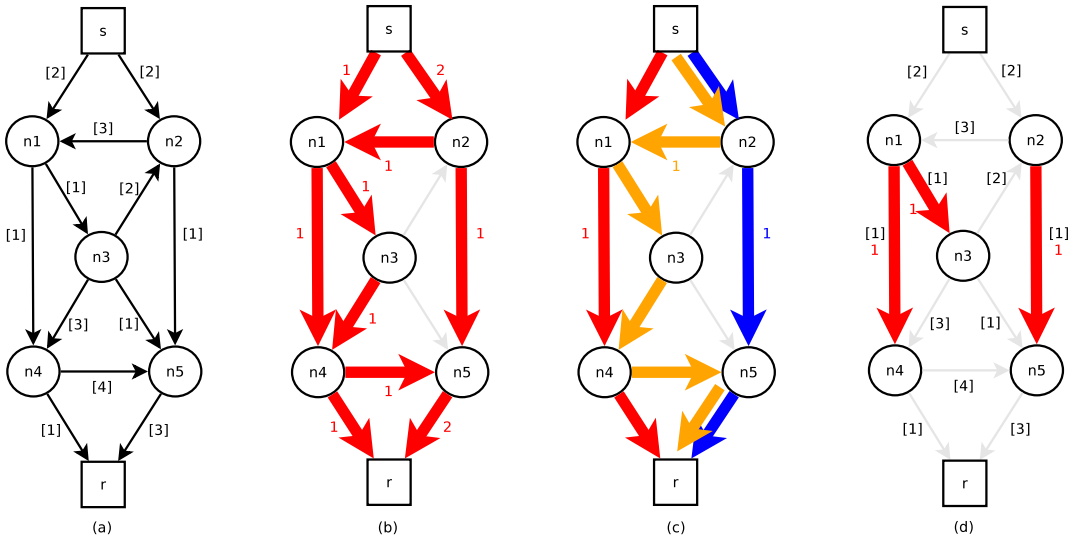


Figure 2.1 – (a) A small directed network. (b) A maximum flow of value 3 in this network. (c) A path decomposition associated to the previous flow. Each path is used to convey 1 unit of flow. (d) The minimum cut induced by the arcs in $\delta^+(U)$ where $U = \{s, n_1, n_2\}$.

2.1.3.7 Summary

Given a telecommunication network with a single source, a single receiver, and a capacity on each channel, the maximum flow problem allows to properly define and compute the maximum amount of information one can theoretically convey from the source to the receiver through the channels of the network while meeting the capacity requirement of each channel. In the following we shall see how the above study can be extended to the multi-receivers case.

2.2 Maximum Steiner flow

2.2.1 Steiner trees

We first introduce some definitions which will be used in the remaining of this thesis. The name *Steiner* comes from Swiss mathematician *Jakob Steiner* (1796–1863). Given an undirected network $G = (V, E)$ and a subset R of *required vertices*, a *Steiner tree* is a tree in G spanning all vertices in R. By a slight abuse of notation, we often identify the tree, regarded as a network, with the set of all its edges. Namely, we write $e \in t$ to mean that a given edge e is part of a tree t (like we do for a path p). Notice that a Steiner tree may also use vertices outside of R, called *Steiner vertices*. Given a directed network $D = (V, A)$ (or a bidirected network $B = (V, L)$) along with a *root vertex* s and a set of *required vertices* R (not containing s), a *directed Steiner tree*, also known as a *Steiner arborescence*, or *Steiner out-branching*, is a directed tree, rooted at s, and spanning all vertices in R. A directed Steiner tree may also use vertices outside of R. For the sake of simplicity, we shall use the term *Steiner tree* regardless of the network structure in the remaining of this thesis. Whether we are actually considering an undirected or a directed Steiner tree can always be inferred from the context. For example, we write $a \in t$ to mean that a given arc a is part of a *directed* Steiner tree. In the following, we will often consider vectors whose components are indexed by the set of all Steiner trees of a given network. Dealing with such a vector requires some cautions, since a network may have a gigantic number of Steiner trees. For example, the undirected network depicted on Figure 2.2 has roughly 50 millions Steiner trees [39].

2.2.2 Problem statement

A telecommunication network is often requested to perform the basic task of conveying the *very same information from a source to a set of terminal nodes*. Assuming there are at least two such nodes, the network operator is naturally prone to save resources, like bandwidth or energy, by grouping those individual requests. One opportunity to do so is by using the so-called *multicast routing*. As explained in the previous chapter, *multicast* is a technique allowing any intermediate node of the network to make copies of its input

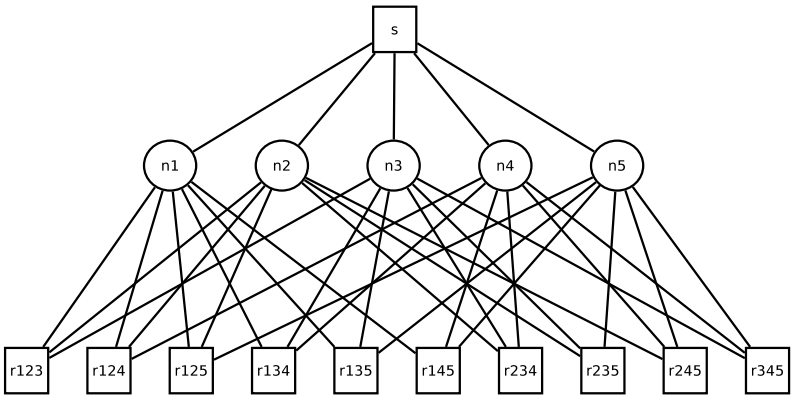


Figure 2.2 – An undirected network with 16 vertices, among which 11 are required (denoted by a square), and 35 edges. This network has 49 956 624 Steiner trees spanning all required vertices [39].

data. The set of edges used to convey a particular piece of data induces a *multicast tree*, or *Steiner tree*, in the network. The model presented below is a *fractional packing of Steiner trees* studied extensively by Jain et al. in [40]. A similar model was proposed by Garg et al. [41] for *overlay multicast*, a setting where only some nodes called *end-hosts* are allowed to duplicate data while other nodes are only forwarding it.

We focus on the case of a directed network since it encompasses the case of a bidirected or undirected one. The instance is made of a directed network $D = (V, A)$ with a *source* s, a set R of *receivers* (not containing s), and a *capacity* c_a for each arc a. We assume that the network is *connected*, so that any receiver can get information from the source. In a directed network, this last assumption implies that there is at least one directed path from the source toward each receiver. A *Steiner flow* x is an assignment of a non-negative real value x_t to each Steiner tree t spanning s and R. For any Steiner tree t, the quantity x_t is called the *amount of flow on tree* t. A Steiner flow x is *feasible* if it satisfies all *capacity requirements*, namely that, for any arc a, the sum of the flow contribution over all trees using a is upper-bounded by the arc capacity c_a. We denote by $\mathcal{X}_\mathcal{T}$ the set of all feasible Steiner flows:

$$\mathcal{X}_\mathcal{T} = \left\{ x \in \mathbb{R}_+^\mathcal{T} : \sum_{t \in \mathcal{T}_a} x_t \leq c_a \, \forall a \in A \right\} \tag{2.20}$$

where $\mathcal{T}$ is the set of all Steiner trees spanning s and R, and $\mathcal{T}_a$ is the subset of such trees using arc a. The *throughput*, or *value*, $|x|$ of Steiner flow x is the overall sum of the flow

contribution of each tree:

$$|x| = \sum_{t \in \mathcal{T}} x_t \tag{2.21}$$

We are now ready to properly define our problem:

Problem	*Maximum Steiner flow*
Instance	Network $D = (V, A)$, source s, set of receivers R, and capacity c_a on each arc a
Solution	A feasible Steiner flow $x \in \mathcal{X}_\mathcal{T}$ in D
Objective	Maximize the value $\|x\|$ of Steiner flow x as defined in Equation (2.21)

We denote by φ_S the value of a maximum Steiner flow for the considered instance:

$$\varphi_S = \max_{x \in \mathcal{X}_\mathcal{T}} \sum_{t \in \mathcal{T}} x_t \tag{2.22}$$

Observe that, when the set R is a singleton, $R = \{r\}$, a Steiner tree spanning s and R is a simple path between the source and the unique receiver r. Hence the maximum Steiner flow problem can be regarded as a generalization of the classical maximum flow problem to more than one receivers.

Another interesting special case arises when all vertices except the source are receivers, $R = V \setminus \{s\}$. Notice that, in this setting, a Steiner tree spanning s and R actually spans all vertices of the network. In this case, all Steiner trees are in fact spanning trees and we can define a *spanning flow* accordingly. We shall refer to the associated problem as the *maximum spanning flow problem*:

Problem	*Maximum spanning flow*
Instance	Network $D = (V, A)$, source s, capacity c_a on each arc a
Solution	A feasible spanning flow $x \in \mathcal{X}_\mathcal{T}$ in G
Objective	Maximize the throughput $\|x\|$ of spanning flow x as defined in Equation (2.21)

We still denote by φ_S the value of a maximum spanning flow for the considered instance, since the meaning should be clear from context.

2.2.3 A detailed example

Consider the three networks depicted on Figure 2.3. Notice that they share the same underlying topology and only differ by the type of their communication channels. In the

following, we shall highlight the influence of the network structure on both the optimal value and the shape of an optimal solution to the maximum Steiner flow problem.

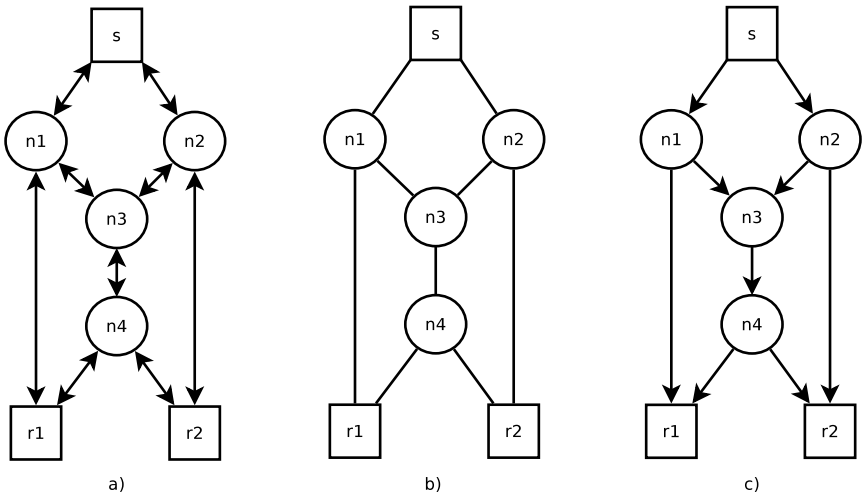


Figure 2.3 – (a) The bidirected butterfly network. (b) The undirected butterfly network. (c) The directed butterfly network. In each case, all channels have a one unit capacity.

2.2.3.1 Steiner flow in the bidirected butterfly network

We first have a look at the bidirected butterfly network. Recall that, in this case, each communication channel is made of two links oriented in opposite directions, each link being equipped with a capacity of its own, and the two capacities sharing the same value. The Steiner flow depicted on Figure 2.4 uses two Steiner trees, each tree conveying 1 unit of flow to both receivers, inducing an overall throughput of value 2. Notice that this Steiner flow satisfies all capacity requirements. We shall prove later that it is a maximum Steiner flow in the bidirected butterfly network.

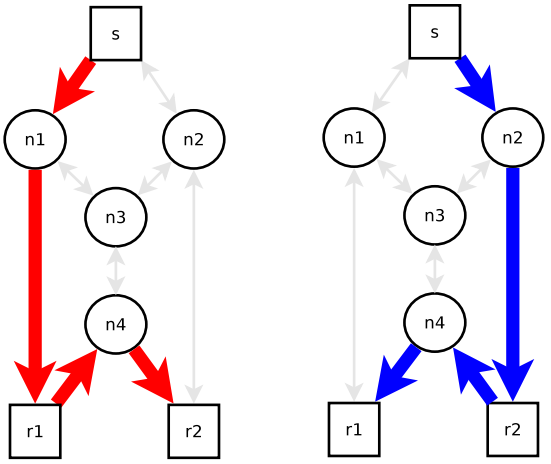


Figure 2.4 – A Steiner flow in the bidirected butterfly network. Each one of those two trees conveys 1 unit of flow to both receivers. Thus this Steiner flow delivers a throughput of value 2.

2.2.3.2 Steiner flow in the undirected butterfly network

We now turn our attention toward the undirected butterfly network where each channel can convey flow in both directions by using the capacity of one single link. Consider the Steiner flow depicted on Figure 2.5. This flow uses nine Steiner trees. Each tree on the first line of Figure 2.5 conveys $\frac{1}{8}$ units of flow while each of the six remaining trees on the second and third lines brings $\frac{1}{4}$ units of flow to both receivers. Thus this Steiner flow provides an overall throughput of value $\frac{15}{8}$. Again, one can check that this Steiner flow is indeed feasible with respect to the capacity constraints, and we shall proof later that it is actually an optimal solution to the maximum Steiner flow problem in the undirected butterfly network.

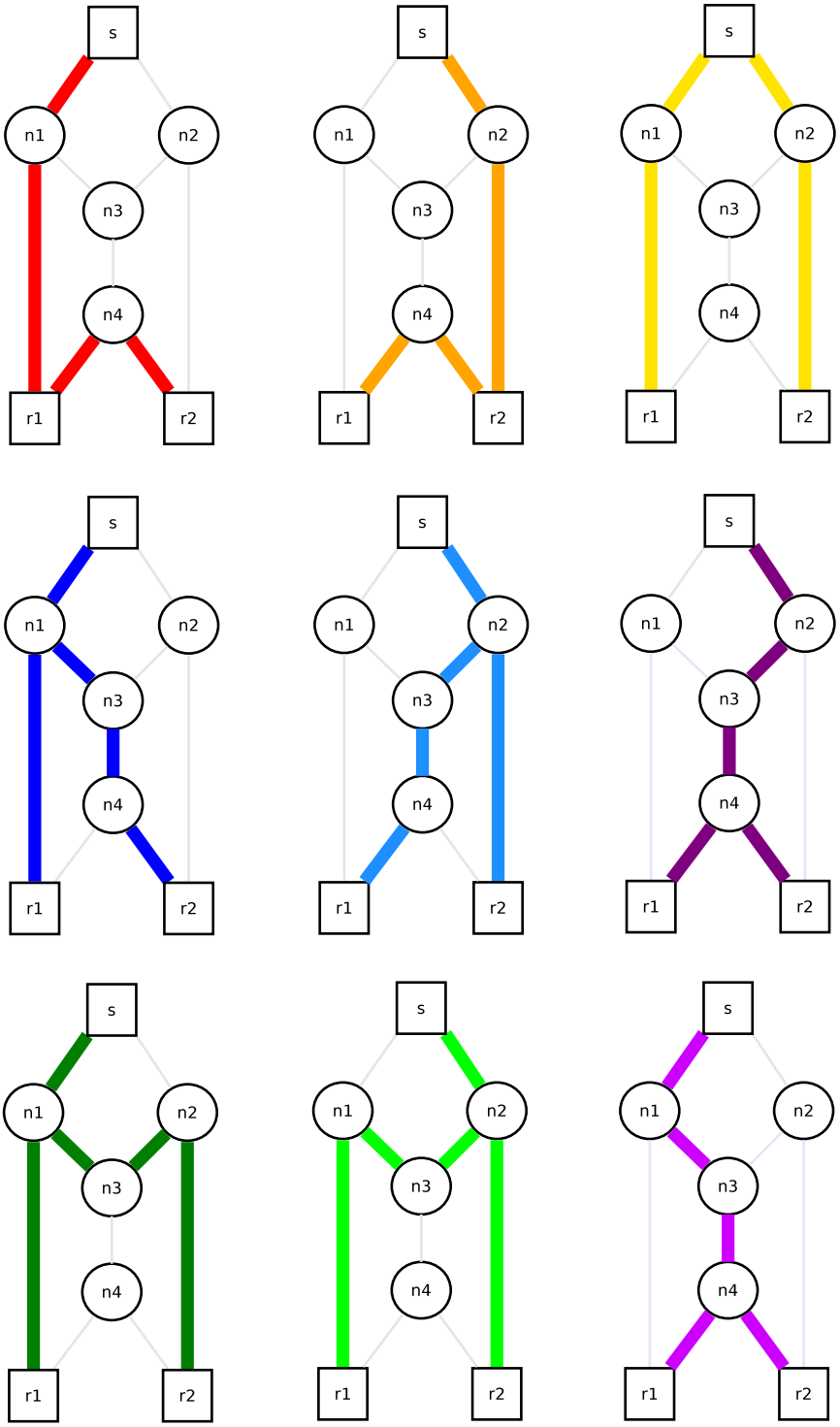


Figure 2.5 – A Steiner flow in the undirected butterfly network. Each tree on the first line conveys $\frac{1}{8}$ units of flow, while each of the six remaining trees below carries $\frac{1}{4}$ units of flow. This Steiner flow ensures that each receiver gets $\frac{15}{8}$ units of flow.

2.2.3.3 Steiner flow in the directed butterfly network

Finally, we consider the directed butterfly network. Figure 2.6 provides a description of a Steiner flow using three Steiner trees, each one carrying $\frac{1}{2}$ units of flow, for an overall throughput of value $\frac{3}{2}$. This Steiner flow satisfies all capacity requirements and we shall justify later its optimality with respect to the maximum Steiner flow problem in the directed butterfly network.

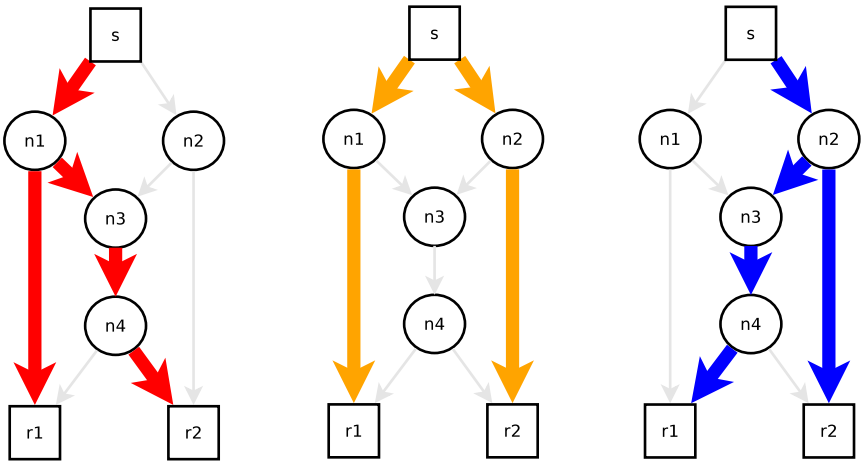


Figure 2.6 – A Steiner flow in the directed butterfly network. Each one of the three trees conveys $\frac{1}{2}$ units of flow, inducing an overall throughput of value $\frac{3}{2}$.

2.2.4 Linear programming formulation

We will now provide a linear programming formulation of the maximum Steiner flow problem, originally presented in [40] as a problem of *fractionally packing* Steiner trees:

$$\begin{cases} \varphi_S = \max \sum_{t \in \mathcal{T}} x_t & (2.23) \\ \text{s.t.} \quad \sum_{t \in \mathcal{T}_a} x_t \leq c_a \quad \forall a \in A & (2.24) \\ \qquad x_t \geq 0 \quad \forall t \in \mathcal{T} & (2.25) \end{cases}$$

where, for each arc a, $\mathcal{T}_a$ is the set of all Steiner trees using a. For each Steiner tree t, variable x_t stands for the amount of flow conveyed by t. Since the number of Steiner trees in a graph may be exponential with respect to its number of arcs or vertices, the number of variables in the above linear program may be exponential in the size of the instance. Notice however that there is only one capacity constraint for each arc of the network, leading to

a polynomial number of Constraints (2.24). Also observe that this linear program is a *fractional packing problem*, also called *positive linear programming*, since all entries of the constraint matrix, constraint vector, and objective vector are non-negative. We denote by y the vector of dual variables associated to Constraints (2.24). The dual of the previous formulation is given by:

$$\begin{cases} \min \sum_{a \in A} c_a y_a & (2.26) \\ \text{s.t.} \quad \sum_{a \in t} y_a \geq 1 \quad \forall t \in \mathcal{T} & (2.27) \\ \qquad y_a \geq 0 \quad \forall a \in A & (2.28) \end{cases}$$

This time we end up with a polynomial number of variables and an exponential number of constraints in this linear program. Notice that the dual of the previous packing problem is a *covering problem*: One wishes to assign a non-negative value to each arc of the network so as to minimize the overall sum of these values, while ensuring that the sum of these values over all arcs of any Steiner tree is at least 1, as described by Constraints (2.27).

2.2.5 Example continued

Back to our previous example with the butterfly networks, we are now ready to justify our assertions regarding the optimality of each Steiner flow presented above. For each network, a solution to the dual of the linear programming formulation of the maximum Steiner flow problem is depicted on Figure 2.7. To certify the feasibility of a proposed solution with respect to Constraints (2.27), one has to check that the sum of the values over all edges of any Steiner tree is at least 1. This can be done easily in both the bidirected and directed cases, while a more careful inspection, involving the consideration of many trees, may be required in the undirected case. For each butterfly network, the dual solution value equals the one of the corresponding Steiner flow depicted above, see Table 2.1. The optimality of those flows hence follows from weak duality in linear programming.

2.2.6 Complexity and algorithms

While talking about complexity, and unless explicitly mentioned, the running time of an algorithm is evaluated by providing an upper-bound on the worst case. This upper-

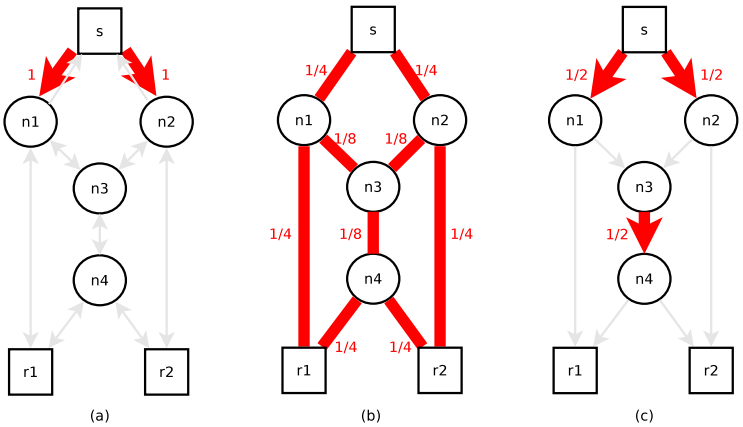


Figure 2.7 – A feasible solution to the dual of the above linear programming formulation of the maximum Steiner flow problem in (a) the bidirected butterfly network, (b) the undirected butterfly network, (c) the directed butterfly network. To avoid any confusion, only positive values are depicted on the figure.

Structure	Throughput	Dual value
Bidirected	2	2
Undirected	$\frac{15}{8}$	$\frac{15}{8}$
Directed	$\frac{3}{2}$	$\frac{3}{2}$

Table 2.1 – For each butterfly network, the throughput of the Steiner flow previously described equals the value of the proposed dual solution.

bound is a function of the *instance size*, namely the network order n, its size m, and the number k of receivers.

A polynomial-time algorithm to solve the maximum Steiner flow problem in a *bidirected network* has been presented in [42]. This algorithm crucially relies on the notion of *arc splitting*, replacing a pair of arcs (u, v) and (v, w) by a single arc (u, w). The main idea is to reduce the initial problem to one of packing spanning arborescences in a suitable network. This new network is obtained from the original one by performing a sequence of splitting operations, so as to isolate any relay node while preserving the global connectivity between the source and any receiver.

It turns out that the complexity of the maximum Steiner flow problem in undirected networks, respectively directed networks, is heavily tied to the complexity of the so-called *minimum Steiner tree problem*, respectively *minimum Steiner arborescence problem*:

Problem	*Minimum Steiner tree*
Instance	Undirected network $G = (V, E)$, set of required vertices R, and weight w_e on each edge e
Solution	A tree t spanning R in G
Objective	Minimizing the overall weight $\sum_{e \in t} w_e$ of tree t

Problem	*Minimum Steiner arborescence*
Instance	Directed network $D = (V, A)$, root vertex s, set of required vertices R, and weight w_a on each arc a
Solution	An arborescence t rooted at s and spanning R in D
Objective	Minimizing the overall weight $\sum_{a \in t} w_a$ of arborescence t

Given a minimization (respectively maximization) problem Π along with a positive real-valued function α, an *approximation algorithm with ratio* α, or α*-approximation algorithm*, is a procedure which finds, for any instance of Π, a feasible solution whose value is at most (respectively at least) α times the value of an optimal solution of Π for this instance, in time polynomial with respect to the size of the considered instance. For the sake of simplicity, we shall assume that the function α takes its value in the interval $[1, +\infty[$ regardless of the optimization direction (minimization or maximization) since the meaning should be clear from the context. The problem Π admits a *constant approximation ratio* if there exists an α-approximation algorithm where α is a constant function. A *polynomial-time approximation scheme (PTAS)* is a parametrized approximation algorithm with ratio $1 + \epsilon$ where the positive parameter ϵ can be made arbitrarily small. A *fully polynomial-time approximation scheme (FPTAS)* is a PTAS which runs in time polynomial in $\frac{1}{\epsilon}$. Finally, notice that a 1-approximation algorithm actually returns an exact solution.

The minimum Steiner tree problem is NP-hard to solve [43], and it is actually NP-hard to approximate it within an approximation ratio of $\frac{96}{95}$ [44]. As far as we know, the algorithm providing the best approximation ratio for solving this problem in polynomial time has been proposed in [45]. This algorithm returns a Steiner tree whose weight is at most 1.39 times the weight of an optimal solution.

The minimum Steiner arborescence problem is also NP-hard, and, given any positive ϵ, it can not be approximated within a ratio better than $O(\log^{2-\epsilon} n)$ unless NP $\subseteq$ ZPTIME($n^{\text{polylog}(n)}$) [46]. However, the minimum Steiner arborescence problem is ap-

proximable in polynomial time within ratio $O(k^\epsilon)$ for any positive ϵ, see [47].

The following theorem, stated in [40], establishes the fundamental connection between the maximum Steiner flow problem and the minimum Steiner tree problem.

Theorem 1. *[40, Theorem 4.1] There is an α-approximation algorithm for the maximum Steiner flow problem in undirected networks, respectively directed networks, if and only if there is an α-approximation algorithm for the minimum cost Steiner tree problem, respectively minimum cost Steiner arborescence problem.*

Theorem 1 means that the approximation ratio is actually preserved when passing from the minimum cost Steiner tree problem to the maximum Steiner flow problem. The proof relies on the following fact: given a candidate solution y to the dual linear program of the maximum Steiner flow problem, the process of testing whether y is actually feasible with respect to Constraints (2.27), is equivalent to solving the so-called *separation problem* of the dual. But this *dual separation problem* is exactly the minimum cost Steiner tree problem with weight y_a on each arc a. We point out to the reader that the proof also rests on the use of the ellipsoid method [48].

Regarding the maximum *spanning* flow problem, the separation problem of its dual is the *minimum spanning tree problem* which can be solved in strongly polynomial time thanks to Kruskal's algorithm [49], Prim's algorithm [50], or even by using a linear programming compact formulation of this problem as the one proposed in [51]. The directed version of this problem, referred to as the *minimum spanning arborescence problem* in the literature, can be solved in strongly polynomial time by the Chu–Liu/Edmonds' algorithm [52, 53]. To directly solve the maximum spanning flow problem, one can use the algorithm proposed by Gabow and Manu in [54] which returns a maximum spanning flow in strongly polynomial time, regardless of the network structure.

By combining this last theorem with the results mentioned above, one immediately gets the content of Table 2.2 regarding complexity and approximability of the maximum Steiner flow problem.

Structure	Complexity	Inapproximability	Approximation
Bidirected	P	N/A	1
Undirected	NP-hard	$\frac{96}{95}$	1.39
Directed	NP-hard	$\Omega(\log^{2-\epsilon} n) \quad \forall \epsilon > 0$	$O(k^{\epsilon}) \quad \forall \epsilon > 0$
Spanning	P	N/A	1

Table 2.2 – Complexity and approximation of the maximum Steiner flow problem.

2.2.7 Computing a maximum Steiner flow by column generation

As already mentioned, one of the main difficulties faced when dealing with the maximum Steiner flow problem is the lack of explicit information regarding the set $\mathcal{T}$ of all Steiner trees in the network. Recall that the cardinality of this set may be huge. When trying to solve this problem, it becomes quite natural to look for a method where useful trees would be generated on the fly by the algorithm, so as to avoid any useless, computationally expensive, enumeration of the trees. A popular technique satisfying this requirement is the so-called *column generation method*. We refer the interested reader to the brief review in Appendix A.1 for a presentation of this technique.

When applying this method to solve the maximum Steiner flow problem, to each component of vector x corresponds a Steiner tree. Initialize the procedure by first looking for a tree t, then define a feasible Steiner flow by routing up to the minimum capacity over all arcs of t on tree t. Notice that solving the pricing problem for a given Steiner flow x actually amounts to finding an *unknown* Steiner tree t maximizing the reduced cost given by:

$$\mathbb{r}_t = 1 - \sum_{a \in t} y_a \tag{2.29}$$

where y is the corresponding dual vector. Hence, one is looking for a Steiner tree t whose overall cost is minimum, given weight y_a on each arc a, which is exactly the minimum cost Steiner tree problem. Although this problem is NP-hard, it can be solved to optimality by using one of its many classical mixed-integer linear programming formulations, see [55]. For our purpose, it will be convenient to use the mixed-integer linear program below in order to compute a minimum-cost Steiner tree, rooted at s and spanning all vertices in R, in a directed network $D = (V, A)$ with weight w_a on each arc a (a valid model for undirected networks can be inferred from this one).

$$
\begin{cases}
\min \quad \sum_{a \in A} w_a x_a & (2.30) \\
\text{s.t.} \quad \sum_{a \in \delta^+(v)} f_a^r - \sum_{a \in \delta^-(v)} f_a^r = b_v^r(1) \quad \forall r \in R, v \in V & (2.31) \\
\qquad f_a^r \leq x_a \quad \forall a \in A, r \in R & (2.32) \\
\qquad f_a^r \geq 0 \quad \forall a \in A, r \in R & (2.33) \\
\qquad x_a \in \{0,1\} \quad \forall a \in A & (2.34)
\end{cases}
$$

where

$$
b_v^r(\phi) = \begin{cases} \phi & \text{if } v = s \quad (2.35) \\ -\phi & \text{if } v = r \quad (2.36) \\ 0 & \text{otherwise} \quad (2.37) \end{cases}
$$

For each arc a, and each receiver r, variable f_a^r is set to 1 if and only if arc a is used to connect source s to receiver r in the tree. For each arc a, variable x_a is set to 1 if and only if a is part of the tree. Constraints (2.31) ensure that the tree interconnects the source and each receiver. Constraints (2.32) prevent an arc a from being used in the tree unless this arc is made part of the tree. The column generation method applied to the maximum Steiner flow problem can be summarized by the scheme depicted on Figure 2.8.

2.2.8 Greedy packing of widest Steiner trees

The following heuristic algorithm for solving the maximum Steiner flow problem was proposed in [56]. The main idea is to perform a greedy packing of Steiner trees. To understand the incentive behind this algorithm, assume one is restricted to use a single Steiner tree to convey flow from the source to all receivers. Since one is interested in maximizing the throughput experienced by any receiver, the most promising tree is the one allowing to convey as much flow as possible. It should be clear that the maximum amount of flow which can be carried by a tree is upper-bounded by the minimum capacity among all edges of this tree. Hence, one is looking for a Steiner tree whose minimum capacity is maximum among all Steiner trees of the network. Such a tree is called a *widest Steiner tree*, or *bottleneck Steiner tree*, and it naturally gives rise to the *widest Steiner tree problem*, also known

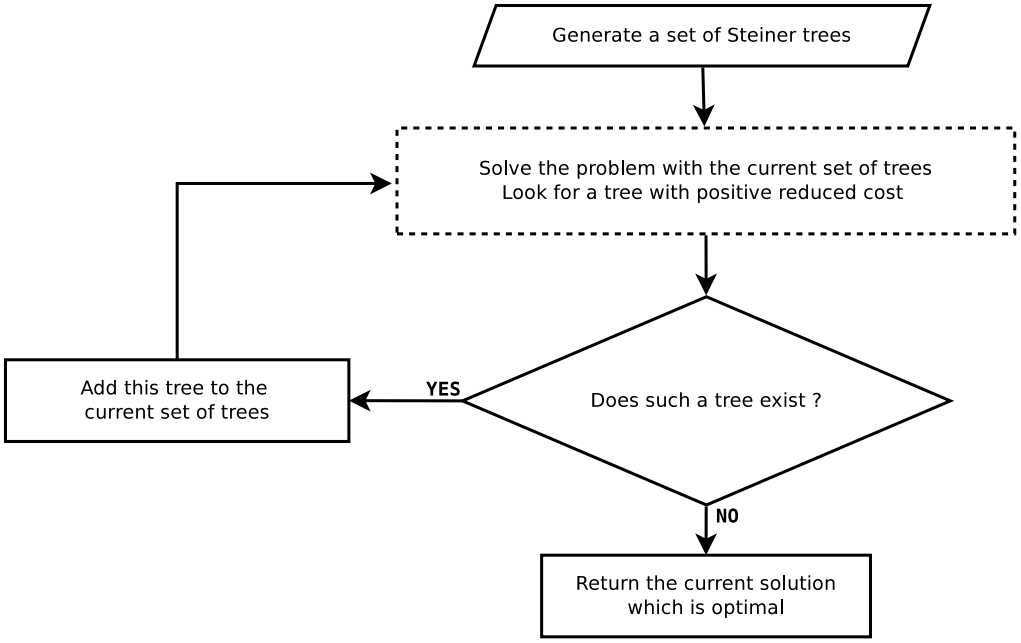


Figure 2.8 – Scheme of the column generation procedure specialized to solve the maximum Steiner flow problem. A mixed-integer linear programming solver may be called so as to get a minimum-cost Steiner tree when applying the algorithm main step (described in the dotted box).

as the *bottleneck Steiner tree problem*.

Problem	*Widest / bottleneck Steiner tree*
Instance	Network $D = (V, A)$, root vertex s, set of required vertices R, and capacity c_a on each arc a
Solution	A tree t rooted at s and spanning R in D
Objective	Maximize the minimum capacity $\min_{a \in t} c_a$ of tree t

Multiple algorithms solving this last problem in polynomial time, with respect to the instance size, have been proposed in the literature, see [56] and references therein. See also the modification of Dijkstra's algorithm presented in [57] to solve the mirror problem of finding a Steiner tree whose maximum weight is minimized in a weighted network. Reversing inequalities in this last algorithm immediately yields a procedure to compute a widest Steiner tree.

Now observe that one can use any algorithm to solve the widest Steiner tree problem as a subroutine in a greedy packing as follows: Starting from the original network, first

call the subroutine to get a widest Steiner tree. Route the maximum feasible amount of flow along this tree and update all residual capacities of the network accordingly. Reiterate the search for a widest Steiner tree in the network with the residual capacities. Continue this process until no new tree can be found. Observe that, at each iteration (except the last one), at least one arc is saturated, hence the whole procedure performs at most one call to the subroutine for each arc of the network. Algorithm 1 provides the corresponding pseudo-code.

Algorithm 1 : algorithm for solving the maximum Steiner flow problem.

Input: Network $D = (V, A)$, source s, set of receivers R, capacity c_a for each arc a;
Output: A set T of Steiner trees and a feasible Steiner flow x using trees in T;

1. Set the residual capacity of arc a equal to the original one, $res[a] = c[a]$;
2. Initialize the set of supporting trees, $T = \emptyset$;
3. **while** There is a widest Steiner tree t with respect to the residual capacities **do**
4. Compute $\nu = \min_{a \in t} res[a]$;
5. Add t to the set of trees, $T = T \cup \{t\}$;
6. Saturate the tree, $x[t] = \nu$;
7. Update the residual capacity of each arc a of t, $res[a] = res[a] - \nu$;
8. **end while**
9. Return x and T.

2.3 Maximum coded flow

2.3.1 Problem statement

Network coding in multicast networks takes its roots in the pioneering work by Ahlswede et al. [15] where the following theorem, referred to as the *fundamental theorem of network coding (in directed multicast networks)*, has first been stated.

Theorem 2. *[15, Theorem 1] Given* directed *network* $D = (V, A)$ *with* source vertex s*, set of* receiver vertices R*, and capacity* c_a *for each arc* a*, the maximum amount of the* same *information which can be simultaneously conveyed in network* D *from* s *to each receiver in* R*, assuming both multicast and network coding are allowed,* equals *the minimum, among all receivers, of the value of a maximum flow between the source and any given receiver.*

The original proof in [15] rests on information theory, see also its presentation by Yeung in the book [16]. An alternative proof relying on linear algebra has been provided by Li

et al. in [58], while a third one grounded on polynomial algebra is due to Koetter and Médard [59]. It should be clear to the reader that, regardless of the routing technique, the amount of information which can be conveyed from the source s to any receiver r is actually upper-bounded by the value of a minimum cut, or equivalently the one of a maximum flow, between the source and this receiver, *even if said receiver is requesting data alone*. Since the goal is to send the same data to all receivers, it is rather natural that the amount of information which can be delivered to all receivers is upper-bounded by the minimum of those values of maximum flows. The breakthrough brought by this last theorem lies in that *this upper-bound can actually be achieved by performing multicast and coding operations in the network*. This last result fundamentally implies that, in order to compute the maximum amount of information one can expect to convey from the source to all receivers in a telecommunication network where both multicast and network coding are simultaneously allowed, it is sufficient to solve a set of *independent* maximum flow problems, where the capacity of any arc of the network can be *fully used by each receiver*, without struggle between receivers over bandwidth consumption of a channel. This last remark is the main incentive for the coded flow model presented below.

The fundamental theorem of network coding holds for both directed and bidirected networks. **Beware however, that this result is *not* valid in any undirected network.** The following counter-example is taken from [60]. Consider the small triangle network depicted on Figure 2.9 a) with a source, two receivers, and a unit capacity on each edge. It should be clear that the value of a minimum cut between the source and each receiver is 2. However, it can be shown that the maximum amount of information the source can provide to both receivers actually equals $\frac{3}{2}$, since it is not possible to convey one unit of data from r_1 to r_2 while *simultaneously* sending another unit from r_2 to r_1. This throughput of value $\frac{3}{2}$ is achievable with data replication alone (without coding), as depicted on Figure 2.9 b). In the following, we will provide a linear program proposed by Li et al. [60, 39] to properly define and compute the maximum amount of the *same* information which can be simultaneously conveyed from a source to a set of receivers, thanks to network coding, in an undirected multicast network G.

For now, we consider a directed or bidirected network and we focus on the setting where

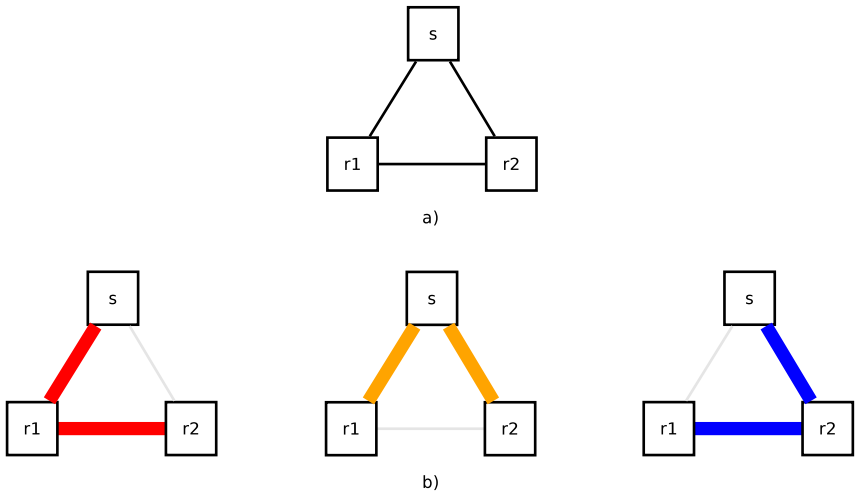


Figure 2.9 – a) A small network made of a source and two receivers connected as a triangle. Each edge has a one unit capacity. b) A Steiner flow in this network using three trees, each one conveying $\frac{1}{2}$ units of flow. Each receiver get $\frac{3}{2}$ units of flow which is also the maximum throughput achievable in this network by using both multicast and network coding. The value of a minimum cut between the source and any receiver is 2.

both copying and coding mechanisms are allowed. The framework is the same as the one of the maximum Steiner flow problem. The instance is still made of a directed network $D = (V, A)$ with source s, set of receivers R (not containing s), and a capacity c_a for each arc a. We still assume the network to be connected (there is at least one path from the source to each receiver), so that any receiver can get information from the source. A *coded flow* x is an assignment of a non-negative real value x_p to each simple path p from the source to any receiver r. The quantity x_p is called the *amount of flow on path* p. Observe that, for each receiver r, the *restriction* of a coded flow x to the set of simple paths from s to r is a classical flow. A coded flow x is *feasible* if it satisfies all *capacity constraints*, namely that, for each receiver r, the classical flow, induced by the restriction of x to the set of simple paths from s to r, meets the capacity requirements of a flow. This last definition means that the bandwidth consumption, induced by a feasible coded flow x over an arc of the network, is obtained by taking the *maximum*, rather than the sum, over all receivers, of the amount of flow going from the source toward each receiver through this arc. We denote by $\mathcal{X}_\mathcal{P}$ the set of all feasible coded flows:

$$\mathcal{X}_\mathcal{P} = \left\{ x \in \mathbb{R}_+^\mathcal{P} : \sum_{p \in \mathcal{P}_a^r} x_p \leq c_a \; \forall a \in A, r \in R \right\} \tag{2.38}$$

where, for any receiver r, $\mathcal{P}_a^r$ is the set of all simple (loop-free) paths between s and r using arc a. For each receiver r, we also denote by $\mathcal{P}^r$ the set of all simple paths between the source and this receiver. Finally, let $\mathcal{P}$ be the set of all simple paths between the source and *any* receiver. The *throughput*, or *value*, $|x|$ of coded flow x is the minimum throughput experienced by any receiver:

$$|x| = \min_{r \in R} \sum_{p \in \mathcal{P}^r} x_p \tag{2.39}$$

We can now formally define our problem:

Problem	*Maximum coded flow*
Instance	Network $D = (V, A)$, source s, set of receivers R, capacity c_a on each arc a
Solution	A feasible coded flow $x \in \mathcal{X}_\mathcal{P}$ in D
Objective	Maximize the value $\|x\|$ of coded flow x as defined in Equation (2.39)

In the following, we denote by φ_C the value of a maximum coded flow for the considered instance:

$$\varphi_C = \max_{x \in \mathcal{X}_\mathcal{P}} \min_{r \in R} \sum_{p \in \mathcal{P}^r} x_p \tag{2.40}$$

Observe that, when the set R is a singleton, the maximum coded flow problem is nothing but the maximum flow problem between the source and the unique receiver.

2.3.2 A detailed example

Consider again the three variants of the butterfly network presented earlier on Figure 2.3. For each of the three cases, the path decomposition of a maximum coded flow is depicted on Figure 2.10. Observe that in all cases the maximum coded flow uses two paths for each receiver, each path conveying 1 unit of flow, for an overall throughput of value 2 experienced by both receivers in each network. Although the value of a maximum coded flow can indeed vary with the kind of network considered, this example illustrates that the structure of a maximum coded flow can be considerably simpler than the one of a maximum Steiner flow. Recall however that, for now, we are only taking the routing aspect into account, leaving any actual coding process, occurring at intermediate nodes of the network, largely uncharted.

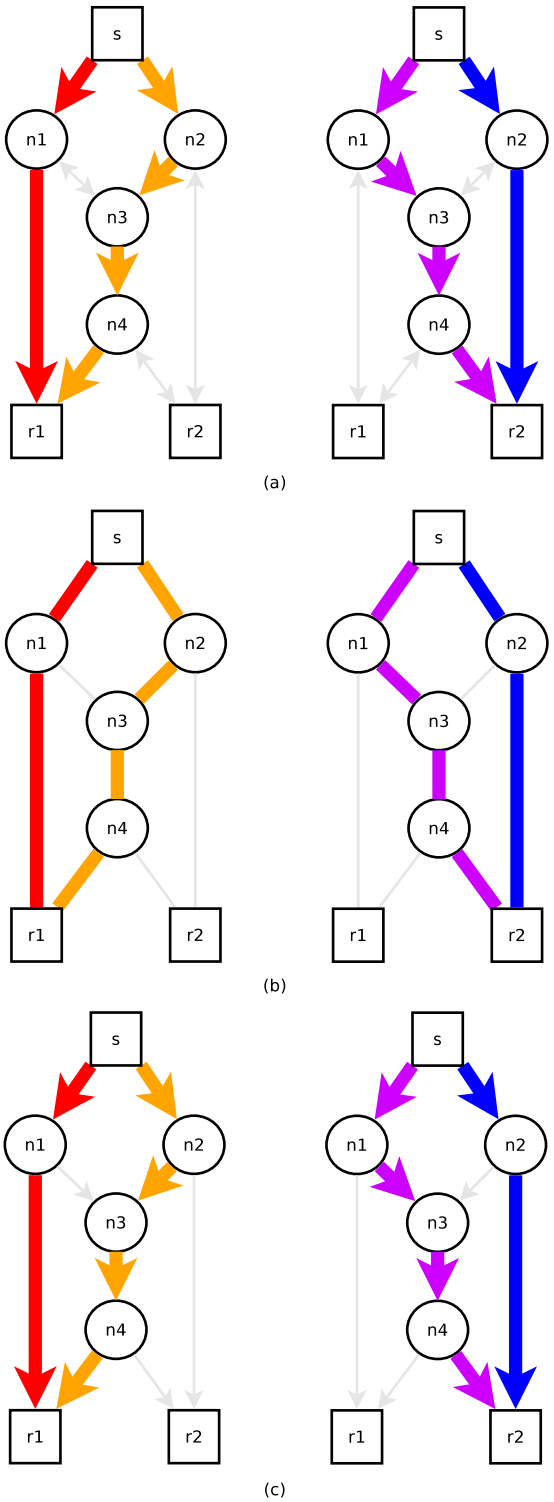


Figure 2.10 – The path decomposition of a maximum coded flow in (a) the bidirected butterfly network, (b) the undirected butterfly network, and (c) the directed butterfly network. In all cases, each path is used to convey 1 unit of flow for a throughput of value 2. The optimality of each coded flow can be inferred by observing that, in each case, the value of a minimum cut from the source to any receiver is 2.

2.3.3 Linear programming formulations

2.3.3.1 Directed and bidirected networks

We shall now provide a linear programming formulation of the maximum coded flow problem in a directed or bidirected network. This formulation is an immediate consequence of Theorem 2:

$$\begin{cases} \varphi_C = \max \varphi & (2.41) \\ \text{s.t.} \quad \varphi \leq \sum_{p \in \mathcal{P}^r} x_p \quad \forall r \in R & (2.42) \\ \quad \sum_{p \in \mathcal{P}_a^r} x_p \leq c_a \quad \forall a \in A, r \in R & (2.43) \\ \quad x_p \geq 0 \quad \forall p \in \mathcal{P} & (2.44) \\ \quad \varphi \in \mathbb{R} & (2.45) \end{cases}$$

For each simple path p, variable x_p stands for the amount of flow conveyed by p. Hence, the number of variables in the above linear program may again be exponential in the instance size. Constraints (2.42) along with the maximization of variable φ ensure that, for any feasible vector x, the throughput experienced by any receiver is at least φ. Constraints (2.43) enforce the network coding capacity requirements. We denote by y the vector of dual variables associated with Constraints (2.43), and by z the one corresponding to Constraints (2.42). The dual of the previous linear program is:

$$\begin{cases} \varphi_C = \min \sum_{a \in A} \sum_{r \in R} c_a y_a^r & (2.46) \\ \text{s.t.} \quad \sum_{r \in R} z^r = 1 & (2.47) \\ \quad \sum_{a \in p} y_a^r \geq z^r \quad \forall r \in R, p \in \mathcal{P}^r & (2.48) \\ \quad y_a^r, z^r \geq 0 \quad \forall a \in A, r \in R & (2.49) \end{cases}$$

Observe that the dual of the problem of finding the maximum flow of minimum value among all receivers consists in looking for a convex combination of cuts, with exactly one such cut induced by each receiver.

We refer to the linear program 2.41 as the *path formulation* of the maximum coded flow problem (in a directed network) so as to distinguish it from the following linear program,

which will be referred to as the *arc formulation* of the same problem:

$$\begin{cases} \varphi_C = \max \varphi & (2.50) \\ \text{s.t.} \quad \sum_{a \in \delta^+(v)} f_a^r - \sum_{a \in \delta^-(v)} f_a^r = b_v^r(\varphi) \quad \forall r \in R, v \in V & (2.51) \\ \qquad f_a^r \leq c_a \quad \forall a \in A, r \in R & (2.52) \\ \qquad f_a^r \geq 0 \quad \forall a \in A, r \in R & (2.53) \\ & (2.54) \end{cases}$$

where

$$b_v^r(\varphi) = \begin{cases} \varphi & \text{if } v = s \quad (2.55) \\ -\varphi & \text{if } v = r \quad (2.56) \\ 0 & \text{otherwise} \quad (2.57) \end{cases}$$

For each arc a and each receiver r, variable f_a^r models the amount of flow going from s to r through a. Constraints (2.51) enforce, for each receiver r, the flow conservation at each vertex of the network of the *individual* flow from s to r. Beware that those flow conservation constraints do *not* impose conservation of the *coded* flow at any vertex of the network. Constraints (2.52) enforce the network coding capacity requirements.

We shall now study the maximum coded flow problem in an undirected network.

2.3.3.2 Undirected networks

As previously mentioned, the network coding fundamental theorem does not hold for undirected networks. However, the following linear program, which is due to Li et al. [60, 39], allows to *define* and *compute* the maximum amount of information which can be conveyed from a source to a set of receivers in a multicast undirected network through the use of network coding techniques. Given an undirected network $G = (V, E)$ with source s, set of receivers R, and capacity c_e for each edge e, the idea is to look for a maximum coded flow in the new instance defined as follows: Consider the bidirected network $B = (V, L)$ obtained by replacing each edge of G by a *pair of reverse arcs*, referred to as a *link*. Keep the source and the receivers as defined in G. Finally, assign a *variable capacity* h_a on each arc a of the bidirected network so that, for each link ℓ, the sum over its two arcs of the variable capacity of each arc should not be greater than the *fixed capacity*

c_ℓ of the edge (in G) associated to this link. Those last requirements, referred to as the *orientation constraints*, ensure that, for each link, the overall amount of flow going through it, taking both directions into account, does not exceed the original capacity of the edge (in the undirected network) associated to this link. Hence, it should be clear that a coded flow meeting those orientation constraints in bidirected network B induces a coded flow in undirected network G satisfying a "reasonable definition" (for a telecommunication practitioner) of the capacity requirements in the undirected network. To summarize the discussion in [60, 39], the following linear program allows to look for a maximum coded flow over all possible orientations of the undirected network:

$$\begin{cases} \varphi_C = \max \varphi & (2.58) \\ \text{s.t.} \quad \varphi \leq \sum_{p \in \mathcal{P}^r} x_p \quad \forall r \in R & (2.59) \\ \quad \sum_{p \in \mathcal{P}_a^r} x_p \leq h_a \quad \forall a \in \ell, \ell \in L, r \in R & (2.60) \\ \quad h_{a_1} + h_{a_2} \leq c_\ell \quad \forall \ell = \{a_1, a_2\} \in L & (2.61) \\ \quad h_a, x_p \geq 0 \quad \forall a \in \ell, \ell \in L, p \in \mathcal{P} & (2.62) \\ & (2.63) \end{cases}$$

For each simple path p, variable x_p stands again for the amount of flow conveyed by p, while, for each arc a, variable h_a models the amount of *coded* flow going through a. Notice that this linear program may also have an exponential number of variables with respect to the instance size. Here also, Constraints (2.59) along with the maximization of variable φ ensure that, for any feasible vector x, the throughput experienced by any receiver is at least φ. Constraints (2.60) can be regarded as capacity requirements with a *variable capacity* h_a for each arc a. Finally, Constraints (2.61) are the previously detailed *orientation constraints*. Observe that, those orientation requirements induce a *flow coupling* between all receivers so that, unlike the previous case, the maximum coded flow problem in an undirected network can not be solved by a decomposition into a set of independent instances of the maximum flow problem. *This also implies that the undirected variant of the maximum coded flow problem can not be reduced to the directed case and hence requires an independent study.* We shall refer to the linear program above as the *path formulation* of the maximum coded flow problem in an undirected network. For practical purposes, it is

often convenient to use the following linear program, which is the original contribution in [60, 39], and will be referred to as the *arc formulation* of the maximum coded flow problem:

$$\begin{cases} \varphi_C = \max \varphi & (2.64) \\ \text{s.t.} \quad \sum_{a \in \delta^+(v)} f_a^r - \sum_{a \in \delta^-(v)} f_a^r = b_v^r(\varphi) \quad \forall r \in R, v \in V & (2.65) \\ f_a^r \leq h_a \quad \forall a \in \ell, \ell \in L, r \in R & (2.66) \\ h_{a_1} + h_{a_2} \leq c_\ell \quad \forall \ell = \{a_1, a_2\} \in L & (2.67) \\ f_a^r, h_a \geq 0 \quad \forall a \in \ell, \ell \in L, r \in R & (2.68) \\ & (2.69) \end{cases}$$

with

$$b_v^r(\varphi) = \begin{cases} \varphi & \text{if } v = s \quad (2.70) \\ -\varphi & \text{if } v = r \quad (2.71) \\ 0 & \text{otherwise} \quad (2.72) \end{cases}$$

For each arc a, and each receiver r of the bidirected network $B = (V, L)$, variable f_a^r models the amount of flow going from s to r through a (as in the case of a directed network). Again, Constraints (2.65) enforce, for each receiver r, the flow conservation at each vertex of the network of the *individual* flow from s to r. Beware that, here also, those flow conservation constraints do *not* impose conservation of the *coded* flow, symbolized by vector h, at any vertex of the network. The other variables and constraints play similar roles in both formulations.

Regarding undirected networks, Li and Li actually show that any exchange of role between the source and a particular receiver leaves the value of a maximum coded flow unaltered [61, Theorem 4].

2.3.4 Complexity and algorithms

As explained above, the achievable capacity of a directed network where multicast and network coding are both simultaneously allowed is exactly the minimum over all receivers of the value of a maximum flow between the source and this receiver. It is then quite natural that computing this value can be done by finding one maximum flow for each receiver. Observe however that we are not only interested in the optimal value φ_C alone

but also in any information regarding the actual routing process. Fortunately, by applying any classical maximum flow algorithm, we can also get a path decomposition for each maximum flow. This implies that we can obtain the supporting paths which shall be used to convey flow to each receiver. We can therefore conclude that solving the maximum coded flow problem can be done in strongly polynomial time by performing one call to a maximum flow subroutine for each receiver of the network.

The previous complexity result relies on the ability to *decompose* the maximum coded flow problem into a set of tractable sub-problems. Since this decomposition property does not hold for each problem considered in this thesis, we shall now present a way to deal with the optimization of a coded flow by a more subtle use of Theorem 2. The next lemma emphasizes the equivalence between the arc and path formulations of the maximum coded flow problem.

Lemma 1. *Consider an instance of the maximum coded flow problem given by a directed network $D = (V, A)$, with source s, set of receivers R, and capacity c_a on each arc a. It is possible to convert, in polynomial time, a basic feasible solution to this instance under one formulation into a basic feasible solution of the same value, for the same instance under the other formulation.*

Proof. It should be clear from the similarity of both formulations that the vectors x and f which collects information regarding each individual flow are the only quantities at stake. Given a basic feasible solution x for the path formulation, a basic feasible solution f for the arc formulation is defined by setting for each arc a and each receiver r:

$$f_a^r = \sum_{p \in \mathcal{P}_a^r} x_p \tag{2.73}$$

This vector f can be computed in polynomial time, provided the number of nonzero components of vector x is itself polynomial with respect to the instance size. Conversely, given a basic feasible solution f for the arc formulation, a basic feasible solution x for the path formulation can be computed as follows: For each receiver r, let f^r be the vector of components f_a^r over each arc a of the network. Observe that vector f^r is the arc projection of a *simple* flow between the source and receiver r. For each receiver r, compute a path decomposition of this simple flow using the algorithm presented by Ahuja et al. [7, Theorem

3.5] which runs in polynomial time. (It is also possible to consider, for each receiver r, the instance of the maximum flow problem defined by setting s as source, r as receiver, and capacity f_a^r on each arc a of the network. The algorithm provided by Edmonds and Karp [38] can compute a path decomposition of a maximum flow for this instance in polynomial time.) The concatenation of all those path decompositions (one per receiver) gives the desired vector x (under the convention that a component of vector x induced by a path which is not returned by the decomposition algorithm is implicitly set to 0). □

Observe that the proof of Lemma 1 rests on an implicit decomposition of the flow and *not* on some proper feature of a coding flow.

Regardless of the network structure (directed or undirected), the arc formulation of the maximum coded flow problem is a linear program involving a polynomial number of continuous variables and constraints, with respect to the instance size. Combined with Lemma 1, this last remark implies that the maximum coded flow problem can be solved in strongly polynomial time in a directed network, even if one is requesting a path decomposition of the solution, as it is often the case in telecommunication.

Although the introduction of another linear program and the use of Lemma 1 may seem cumbersome compared with Theorem 2, this last approach is more general, encompassing the case of undirected networks, since it does *not* rely on the ability to decompose the original problem into a set of tractable sub-problems.

The maximum coded flow problem in an undirected network can also be solved in polynomial time thanks to a distributed algorithm proposed by Li and Li [62].

2.3.5 Coding scheme

The pioneering work of [15] only establishes a theoretical proof of the existence of a coding scheme allowing an operator to meet the maximum network rate achievable through coding. However, their definition of a code is sufficiently generic to encompass many potential coding processes which may be used in practical applications.

The breakthrough paper by Li et al. [58] introduces the concept of *linear network code*, where the output of any node is a linear combination of its inputs with coefficients taken

in a large enough finite field, and highlights that, for a directed acyclic network, such a code is sufficient to achieve the maximum network coding rate. The algebraic framework developed by Koetter and Médard [59] also considers linear network codes over finite fields although their elegant approach rather rests on a subtle combination of algebraic geometry and matrix theory. This framework, further developed in [63], ultimately leads to the idea of random linear network coding, where coefficients in the finite field are randomly picked at each node, so that any receiver can decode its input data with high probability, see the paper by Ho et al. [64]. Random linear network coding is a simple decentralized approach to the network code design problem. Following a long line of research, Jaggi et al. proposed a deterministic polynomial-time algorithm returning a linear network code sufficient to achieve the network coding maximum rate [65].

This last algorithm requires an alphabet of size $O(|R|)$ to encode messages, where R is the set of receivers requesting the data. In [24] Lehman and Lehman exhibit instances where an alphabet of size $\Omega(\sqrt{|R|})$ is required, even if nonlinear network codes are allowed. They also show in the same paper that minimizing the alphabet size of a linear network code is an NP-hard task.

Another practical limitation regarding network coding lies in the number of nodes performing coding operations instead of merely forwarding data. This issue is studied by Langberg et al. in [66], where the authors present an algorithm, improving over the complexity of the one proposed in [65], while providing an upper-bound on the number of nodes involved in the coding process. In the same paper, the authors also show that looking for a network code minimizing the number of coding nodes is an NP-hard problem. Nonetheless, this last problem can be practically tackled by using a heuristic algorithm, as the one proposed by Kim et al. in [67].

Regarding networks with directed cycles, a network decomposition originally proposed by Fragouli and Soljanin [68] suggested a connection between network coding and convolutional codes, see [69] by the same authors. This connection gave rise to a now well-established network coding theory for networks with directed cycles, see the study by Jaggi et al. [70], the one by Li and Yeung [71], and the paper of Erez and Feder [72].

To put an end to our journey through this topic, we would like to mention to the reader

a survey on network coding by Sanna and Izquierdo [73] along with one by Bassoli et al. [20].

2.4 Features of the information flows

2.4.1 Coding gain

One of the primary motivations behind the introduction of coding process at intermediate nodes of a multicast network has been the hope for an improvement of the maximum amount of information the network operator could simultaneously provide to each receiver [15]. Researchers in the network coding community have been pondering about the advantage of deploying such coding process.

Question 1. *Is there an incentive, with respect to the maximum achievable throughput, to use network coding in a multicast network?*

To answer this question one needs a proper criterion to evaluate the benefit of deploying coding techniques inside a network. The following quantity has been retained by the community:

Definition 1. *Given a network $D = (V, A)$ with source s, set of receivers R not containing s, and capacity c_a for each arc a, the* coding gain *g_φ, also known as the* coding advantage*, provided by network coding over multicast alone, is the ratio of the value of a maximum coded flow over the one of a maximum Steiner flow, namely:*

$$g_\varphi = \frac{\varphi_C}{\varphi_S} \tag{2.74}$$

Recall that we only consider networks where there is at least one path between the source and each receiver. Hence, we can safely assume $\varphi_S > 0$ since there exists at least one Steiner tree to convey flow in the network. Thus, the previous definition of the *coding gain* g_φ is well-grounded. A value of g_φ lower than 1 means that allowing coding process to take place inside the network is detrimental to the multicast framework. A value of g_φ greater than 1 implies that the corresponding multicast network actually benefits from the deployment of network coding techniques. Finally a value of g_φ exactly equal to 1 means

that there is no incentive to implement network coding process in the considered multicast network, with respect to this criterion. The following theorem, which is the formalization of a common knowledge in the network coding literature, basically states that allowing network coding deployment inside a multicast network never impedes the performance of this network.

Theorem 3. *Given a network $D = (V, A)$ with source s, set of receivers R, and capacity c_a for each arc a, the coding gain g_φ is at least* 1.

Proof. The underlying idea of this theorem is that routing using multicast only corresponds to the special case of using both multicast and network coding simultaneously in a network where no coding operation is performed, only simple data duplication. Let $\mathbb{x}$ be a maximum Steiner flow. We shall build a feasible coded flow $\mathbb{x}$ with the same throughput as $\mathbb{x}$. For each receiver r and each path p between s and r, define:

$$\mathbb{x}_p = \sum_{\substack{t \in \mathcal{T} \\ p_t^r = p}} \mathbb{x}_t \tag{2.75}$$

where, for any Steiner tree t and any receiver r, p_t^r is the unique path from s to r in t. We first show that coded flow $\mathbb{x}$ satisfies all capacity requirements assuming Steiner flow $\mathbb{x}$ does. Observe that for any arc a and any receiver r:

$$\begin{aligned}
\sum_{p \in \mathcal{P}_a^r} \mathbb{x}_p &= \sum_{p \in \mathcal{P}_a^r} \sum_{\substack{t \in \mathcal{T} \\ p_t^r = p}} \mathbb{x}_t && (2.76)\\
&\leq \sum_{t \in \mathcal{T}_a} \sum_{\substack{p \in \mathcal{P} \\ p_t^r = p}} \mathbb{x}_t && (2.77)\\
&= \sum_{t \in \mathcal{T}_a} \mathbb{x}_t && (2.78)\\
&\leq c_a && (2.79)
\end{aligned}$$

where the first inequality stands since, if arc a is used by path p while p is part of Steiner tree t, then a is also used by t. The last equality holds because there is only one path p_t^r between source s and receiver r in any Steiner tree t. Finally recall that we assume $\mathbb{x}$ to be a feasible Steiner flow from which we get the last inequality. Thus coded flow $\mathbb{x}$ meets

all capacity requirements. Regarding the value of χ, one has for any receiver r:

$$\sum_{p \in \mathcal{P}^r} \chi_p = \sum_{p \in \mathcal{P}^r} \sum_{\substack{t \in \mathcal{T} \\ p_t^r = p}} \aleph_t \tag{2.80}$$

$$= \sum_{t \in \mathcal{T}} \sum_{\substack{p \in \mathcal{P}^r \\ p_t^r = p}} \aleph_t \tag{2.81}$$

$$= \sum_{t \in \mathcal{T}} \aleph_t \tag{2.82}$$

where the last equality comes again from the uniqueness of the path p_t^r for a given Steiner tree t and a given receiver r. By combining this last result with the feasibility of coded flow χ, and the optimality of Steiner flow $\aleph$, one gets $\varphi_S \leq \varphi_C$, or equivalently $g_\varphi \geq 1$. □

2.4.2 Example continued

Consider again the three variants of the butterfly network described earlier. Since we know the value φ_S of a maximum Steiner flow along with the value φ_C of a maximum coded flow, we immediately deduce the value of the coding gain g_φ for each variant, as summarized in Table 2.3. Observe that there is no incentive to use network coding in the bidirected butterfly network, while there is a benefit to perform coding operations in both the undirected and directed butterfly networks. Those results highlight the importance of the network structure on the benefit one can expect to get from network coding.

Structure	φ_S	φ_C	g_φ
Bidirected	2	2	1
Undirected	$\frac{15}{8}$	2	$\frac{16}{15}$
Directed	$\frac{3}{2}$	2	$\frac{4}{3}$

Table 2.3 – The coding gain g_φ of each butterfly network.

2.4.3 Features of the coding gain

We shall now present various results regarding the coding gain g_φ. Most of those results are well-known in the network coding literature.

2.4.3.1 Two special cases

We first have a look at the so-called *unicast setting* where there is only one receiver, $R = \{r\}$. As previously mentioned, in this particular setting, a Steiner tree spanning both the source s and the receiver r is simply a path between s and r. A maximum Steiner flow then becomes nothing but a classical maximum flow under path decomposition. Furthermore, a maximum coded flow also becomes a simple maximum flow since there is only one receiver. The following lemma is thus common knowledge in the network coding literature.

Lemma 2. *[61, Corollary 1] Given a network $D = (V, A)$ with source s, receiver r distinct from s, and capacity c_a for each arc a, it is possible to achieve the maximum throughput between s and r by performing forwarding operations only. Therefore one has $g_\varphi = 1$ in this setting.*

Another interesting setting lies in the other extreme case regarding receivers, the so-called *broadcast setting* where each vertex of the network except the source is a receiver, $R = V \setminus \{s\}$. The following theorem emphasizes that the value of a maximum *spanning* flow equals the one of a maximum coded flow in any directed network. This means that performing coding operations is not required in a directed broadcast network as far as maximum throughput is concerned.

Theorem 4. *[42] Given a directed network $D = (V, A)$ with source s, set of receivers $V \setminus \{s\}$, and capacity c_a for each arc a, the value of a maximum spanning flow and the one of a maximum coded flow are equal, thus $g_\varphi = 1$.*

The proof of this theorem relies on a theorem due to Edmonds on packing arc-disjoint spanning arborescences in a digraph [74], see [75] for details.

The previous result regarding equality between the value of a maximum spanning flow and the one of a maximum coded flow also holds in any undirected network. This leads to the same conclusion as the previous one regarding the potential benefit of network coding in an undirected broadcast network [61, 76].

Theorem 5. *[61, Corollary 2] Given an undirected network $G = (V, E)$ with source s, set of receivers $V \setminus \{s\}$, and capacity c_e for each edge e, the value of a maximum spanning flow and the one of a maximum coded flow are equal, thus $g_\varphi = 1$.*

The proof of this theorem as stated in [61, 76] rests on the celebrated Tutte-Nash-Williams theorem [77, 78], see [75].

2.4.3.2 Bidirected networks

We shall now turn our attention to the coding gain in bidirected networks. The next theorem states that the value of a maximum Steiner flow equals the one of a maximum coded flow in any bidirected network. This means that allowing network coding over multicast in a bidirected network cannot improve the maximum throughput experienced by all receivers [42, 79].

Theorem 6. *[42, Theorem 1] Given a bidirected network $B = (V, L)$ with source s, set of receivers R, and capacity c_a for each arc a, the value of a maximum Steiner flow and the one of a maximum coded flow are equal, namely $g_\varphi = 1$.*

The proof of this theorem relies on the idea of arc splitting, i.e. replacing a pair of arcs (u, v) and (v, w) by a single arc (u, w). Starting from the original network, perform a sequence of arc splitting operations thanks to a theorem due to Frank and Jackson, see [42, 3], so as to obtain a new digraph where all intermediate nodes are isolated. The remaining of the proof involves the use of Theorem 4 to conclude. This theorem is important since the core network of the Internet can be roughly modeled by a bidirected network, see the discussion on this topic in [42, 79] and references therein.

2.4.3.3 Undirected networks

We shall now study the coding gain in undirected networks. The next theorem emphasizes that, in any undirected network, the value of a maximum coded flow is upper-bounded by twice the one of a maximum Steiner flow. This implies that allowing coding operations to take place in an undirected network may at most double the throughput experienced by any receiver.

Theorem 7. *[61, Corollary 2] Given an undirected network $G = (V, E)$ with source s, set of receivers R not containing s, and capacity c_e for each edge e, the value of a maximum coded flow is at most twice the one of a maximum Steiner flow, equivalently $g_\varphi \leq 2$.*

The proof of Theorem 7 is quite similar to the one of Theorem 6 and rests on the idea of edge splitting, replacing a pair of edges $\{u, v\}$ and $\{v, w\}$ by a single edge $\{u, w\}$. Perform a sequence of edge splitting operations thanks to a theorem of Mader, see [61, 80], so as to transform the original network where each capacity has been doubled into a new undirected graph where all intermediate nodes are isolated. The proof is then established thanks to Theorem 5. We would like to point out that alternative proofs of this theorem have been proposed in the literature, resting on either linear programming [81] or graph theory [82].

A quite natural question arising from this last result is whether this upper-bound of 2 on the coding gain of any undirected network is actually tight. To the best of our knowledge, the currently highest example of coding gain is a limit case of $\frac{8}{7}$ [81].

2.4.3.4 Directed networks

We shall now deal with the coding gain in directed networks. The following theorem highlights that, unlike their bidirected or undirected counterpart, those networks can admit arbitrarily large coding gain.

Theorem 8. *[83, 84] For any real number $\eta \geq 1$, there exists an infinite family of directed networks with unit capacity such that the ratio between the value of a maximum coded flow and the one of a maximum Steiner flow satisfies $g_\varphi \geq \eta$.*

The proofs proposed in [83, 84] rely on a family of networks known as *combination networks* in the literature. We denote by $[n]$ the set of all positive integers between 1 and positive integer n (included). Given two positive integers k and n with $k \in [n]$, the directed combination network $C(n, k)$ is made of three layers of vertices. The first layer is made of the source s, the second layer consists of a set of n intermediate nodes identified with the set $[n]$, and the third layer is made of a set R of $\binom{k}{n}$ receivers. We identify each receiver with a subset of $[n]$ of cardinality k. Those three layers are connected together by one arc

from the source to each intermediate node, and one arc from an intermediate node to each receiver which contains this node, regarded as an element of $[n]$. Furthermore each arc a has a unit capacity: $c_a = 1$. Three combination networks are depicted on Figure 2.11.

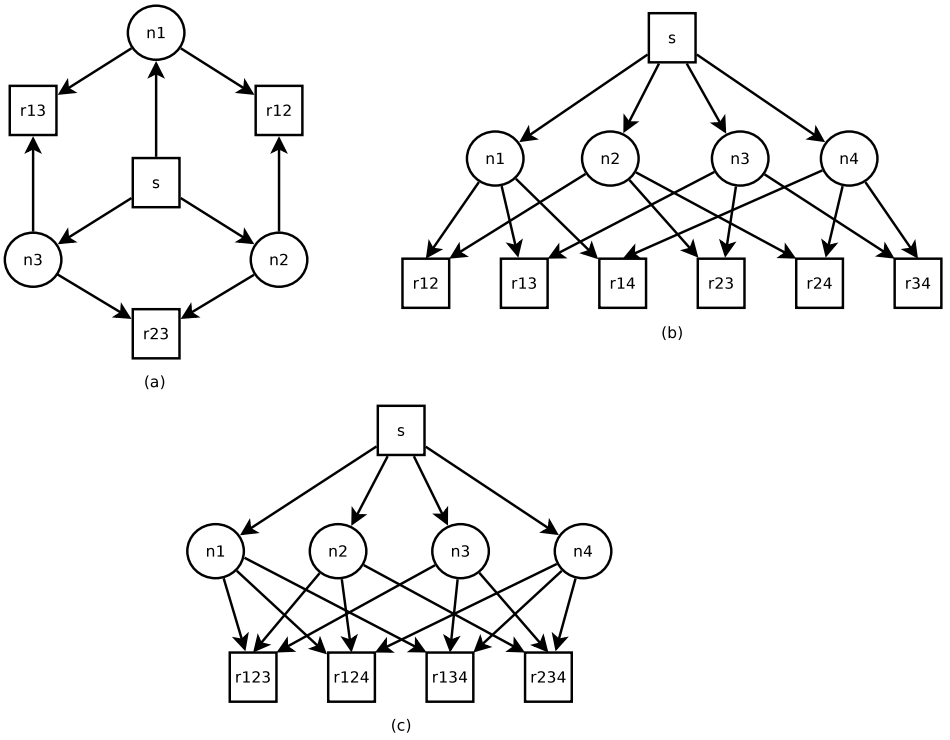


Figure 2.11 – (a) The combination network $C(n, k)$ for $n = 3$ and $k = 2$. (b) The combination network $C(n, k)$ for $n = 4$ and $k = 2$. (c) The combination network $C(n, k)$ for $n = 4$ and $k = 3$.

The proof of Theorem 8 in [83] rests on coding theory while the one in [84] relies on information theory. We shall now give a third original proof that makes use of the theory of duality in linear programming. To do so, we will need the following lemma:

Lemma 3. *In a directed combination network $C(n, k)$, any Steiner tree t rooted at the source s and spanning all receivers R is required to use at least $n - k + 1$ intermediate nodes, or equivalently $n - k + 1$ arcs between the first and second layers.*

Proof. Try to build a Steiner tree. By picking an intermediate node, the tree can span up to $\binom{n-1}{k-1}$ receivers. Thus $\binom{n}{k} - \binom{n-1}{k-1} = \binom{n-1}{k}$ receivers remains unspanned. By picking another intermediate node, at most $\binom{n-2}{k-1}$ new receivers can become part of the tree and

$\binom{n-1}{k} - \binom{n-2}{k-1} = \binom{n-2}{k}$ receivers remains unspanned. By induction we get:

$$\binom{n}{k} = \sum_{j=k-1}^{n-1} \binom{j}{k-1} \tag{2.83}$$

and spanning all receivers can be done by picking $(n-1)-(k-1)+1 = n-k+1$ intermediate nodes but no less. □

We are now ready for the proof of Theorem 8 which rests on arguments similar to those developed in [85] for undirected combination networks.

Proof. Consider a combination network $C(n,k)$. Since there are exactly k arc-disjoint paths from the source to each receiver, one immediately gets the value of a maximum coded flow in this network, $\varphi_C = k$. The next step is to determine the value of a maximum Steiner flow in this network, namely $\varphi_S = \frac{n}{n-k+1}$. Consider the set of n Steiner trees defined as follows: For each integer $i \in [n]$, tree t_i is made of $n-k+1$ intermediate nodes starting from node i up to node $i+n-k$ where each index is taken modulo n. Pick all arcs from the source to those $n-k+1$ intermediate nodes to connect the first and second layers. In order to connect the second and the third layer, start by picking all receivers reachable from intermediate node i, then pick all unspanned receivers reachable from intermediate node $i+1$ (index modulo n) and proceed iteratively by greedily picking all receivers still unspanned. Observe that any arc of the network is used by at most $n-k+1$ trees. This implies that the Steiner flow defined by routing $\frac{1}{n-k+1}$ units of flow on each of those trees and nothing on any other tree meets all capacity requirements. Since its value is $\frac{n}{n-k+1}$, we get $\varphi_S \geq \frac{n}{n-k+1}$. The Steiner flow hence defined for network $C(4,2)$ is depicted on Figure 2.12 (a). Let y be the vector defined by setting $y_a = \frac{1}{n-k+1}$ for each arc from the source to an intermediate node, and $y_a = 0$ for any arc from an intermediate node to a receiver. From Lemma 3, any Steiner tree uses at least $n-k+1$ arcs from the source to intermediate nodes, therefore vector y is a feasible solution to the dual of the maximum Steiner flow problem with value $\frac{n}{n-k+1}$. This feasible dual solution for network $C(4,2)$ is depicted on Figure 2.12 (b). By weak duality in linear programming combined with the previous result we get $\varphi_S = \frac{n}{n-k+1}$. Hence, the coding gain of network $C(n,k)$ is $g_\varphi = \frac{k(n-k+1)}{n}$. Setting $n = 2k$ leads to a coding gain of value $g_\varphi = \frac{k+1}{2}$. This last quantity

is unbounded when the parameter k grows to infinity. Finally, given real number $\eta \geq 1$, the network $C(2k, k)$ with $k = \lceil 2\eta - 1 \rceil$ satisfies all requirements of the theorem. □

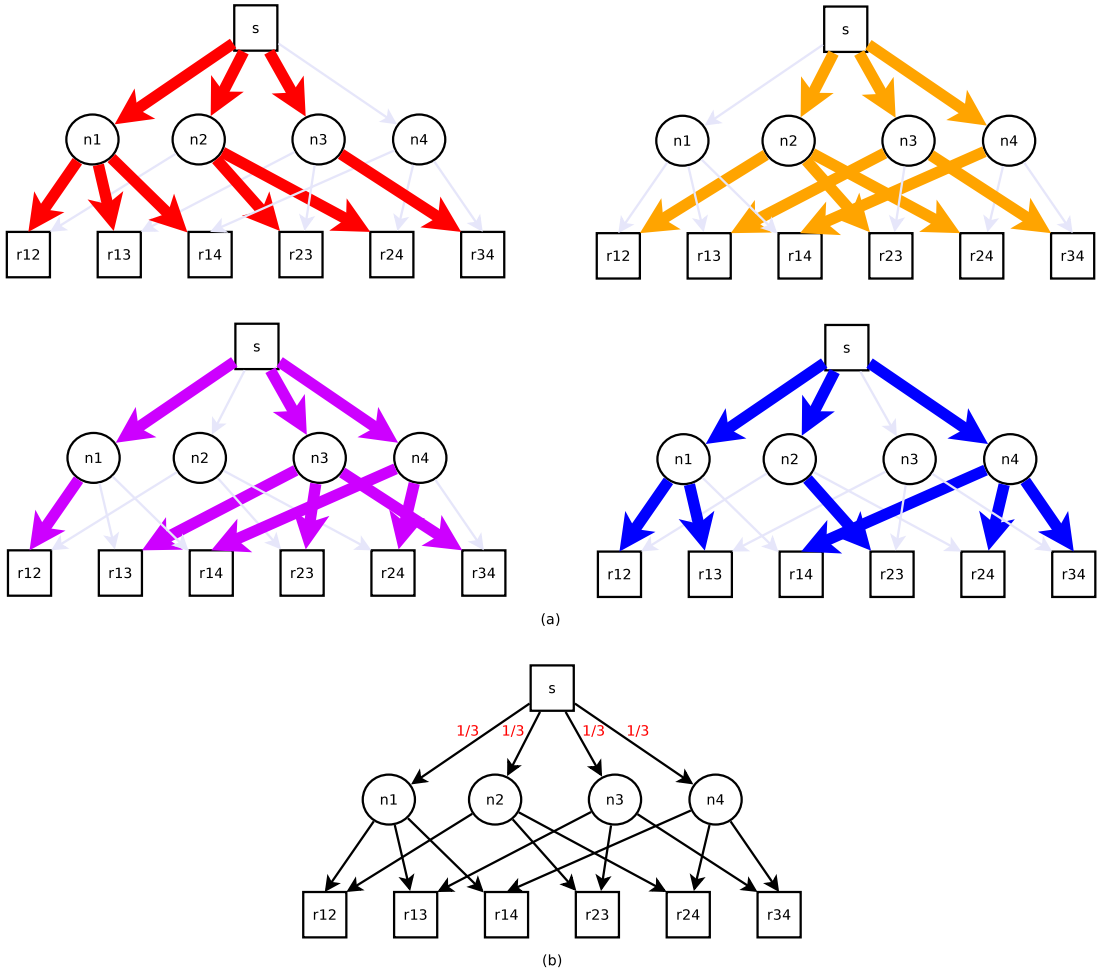


Figure 2.12 – (a) A maximum Steiner flow in network $C(4, 2)$. Each of the four trees conveys $\frac{1}{3}$ units of flow for an overall throughput of value $\frac{4}{3}$. (b) A feasible solution to the dual of the maximum Steiner flow problem. Only arcs from the source to intermediate nodes have a positive value.

We point out that at least one other family of directed networks, first defined in [86], also exhibits provably high coding gain, see [87].

2.4.3.5 Experimental evaluation of the coding gain

All results presented so far involve very structured networks. Regarding the coding gain, one may legitimately ask whether a more realistic network topology could benefit from

network coding, assuming multicast is always allowed. Unfortunately for network coding promoters, it seems that networks for which the coding gain g_φ is strictly greater than one are rather scarce. For undirected networks, the authors of [39] report that on a sample of over one thousand randomly generated topologies, the coding gain value remains stuck to 1.0 in *all* instances. When considering directed networks, a comparison of network coding with various heuristic algorithms solving the maximum Steiner flow problem, including the one presented above, performed on six network topologies of Internet Service Providers, shows few benefits of network coding as far as maximum throughput is concerned [56]. This trend is confirmed by the results presented in [88] where the coding gain remains stuck at 1.0 for hundreds of randomly generated instances.

However, one should not conclude from those results that network coding is devoid of any practical interest. First notice that, in all cases, the computational complexity involved in finding a maximum coded flow is significantly lower than the one required to obtain a maximum Steiner flow. Besides, as already pointed out in [88], the maximum Steiner flow problem is a rather theoretical view of the rate one could expect from the use of multicast techniques. This is because each multicast tree requires constant management from the network operator. Thus the number of trees one could use to convey information through a network is strongly restricted. Adding this constraint on the number of trees when looking for a maximum Steiner flow causes a soaring of the experimental coding gain [88]. Hence, in the mind of a telecommunication operator, network coding should be regarded as a tool allowing one to *practically* achieve the maximum rate of a multicast network, rather than a technique to improve this maximum rate [56, 39].

2.5 Conclusion

In this chapter we presented two network optimization problems, motivated by practical telecommunication applications, namely the maximum Steiner flow problem and the maximum coded flow problem. Since each problem can be used to evaluate the maximum amount of data one can convey from a source node to a set of receiver nodes in a particular setting, multicast alone for the maximum Steiner flow problem, multicast combined with coding for the maximum coded flow problem, one can use both problems to define

an indicator, the coding gain, allowing comparisons between those two settings. We then provided a summary of various and important results which have been previously proposed in the network coding literature, regarding the *domain* of values one can expect the coding gain to take. Those results are summarized in Table 2.4 below. Recall that the greater the value actually taken by the coding gain in a given multicast network, the more beneficial network coding techniques would be if implemented in this particular network.

Structure	Coding gain
Bidirected	$\{1\}$
Undirected	$[1, 2]$
Directed	$[1, +\infty[$

Table 2.4 – Coding gain domain for each channel model.

Table 2.4 also highlights how impactful the channel model actually is on the coding gain. No benefit could be expected from using coding techniques in a bidirected network, while network coding can at most double the maximum amount of data experienced by any receiver in an undirected multicast network. Finally one can expect the coding gain of a directed network to be arbitrarily large.

The approach taken in this chapter will be our reference for the following ones. We shall consider a problem of general interest in the telecommunication community, then propose a network optimization model for the associated problem in each setting, multicast alone or combined with coding techniques. Once done, we shall formally define an associated gain by taking the ratio of their optimal values. We shall then study this ratio in an attempt to evaluate the benefit of introducing coding mechanisms in a multicast network.

Chapter 3

Multicommodity information flows

3.1 Introduction

3.1.1 Motivation

To the best of our knowledge, most of the work in the multicast and network coding literature focuses on the *single commodity case* where one wishes to send the very same information from one source to a set of receivers by performing replication and coding operations at intermediate nodes of the network. However, in practical applications, it may be the case that the network has to deal with a set of sources, each such source wanting to convey its own data to its own set of receivers through the network. In the following, we call *commodity* such a couple (source, set of receivers). The word *commodity* is chosen to highlight the connection with some *multicommodity flow problems* in network optimization [7]. Notice however that the word *session* is frequently used as a synonym of commodity in the field of telecommunications. Although each commodity may be willing to coordinate its efforts with the others, by sharing communication protocols, it is unlikely that one commodity would allow its own data to be mixed with the information flow of another commodity, due to confidentiality issues. This motivates the development of models for handling this *multicommodity* setting we just described, which is the topic of the present chapter.

3.1.2 Content

We shall first provide a brief summary of two classical problems as they appear in the multicommodity flow literature [7], namely the *maximum weighted multicommodity flow problem* and the *maximum concurrent flow problem*. As we did in Chapter 2, we shall then present an extension of each problem in the framework of information flows. We first introduce the notion of multicommodity Steiner flow before studying the *maximum weighted multicommodity Steiner flow problem* and the *maximum concurrent Steiner flow problem*. Roughly speaking, each problem is again obtained by replacing paths with Steiner trees as flow support in a suitable framework. It is then natural to consider the variant of each of the two aforementioned Steiner flow problems where coding mechanisms are allowed to take place in the telecommunication network. We thus successively present the *maximum weighted multicommodity coded flow problem* and then the *maximum concurrent coded flow problem*. Finally, we shall define and study a natural extension of the *coding gain* to the multicommodity setting in an attempt to evaluate the impact of introducing coding techniques in a multicast network. In doing so, we explicitly draw a connection between the coding gain and the newly defined *multicommodity coding gain*. The scheme depicted on Figure 3.1 provides an overview of all problems studied in both the previous and the present chapters along with the interconnections between them. The reader already familiar with the *maximum multicommodity flow problems* may skip the remaining of this introduction. Furthermore, an extensive review of all problems presented below may be unnecessary at first reading.

3.1.3 Multicommodity flows

3.1.3.1 Setting

A multicommodity flow can be regarded as a natural extension of a simple flow (as defined at the beginning of the previous chapter) where the network is requested to convey flow between several couples source/destination. For brevity, we focus on the case of directed networks, without loss of generality since it encompasses the case of undirected ones. Consider a directed network $D = (V, A)$ with a capacity c_a for each arc a. A

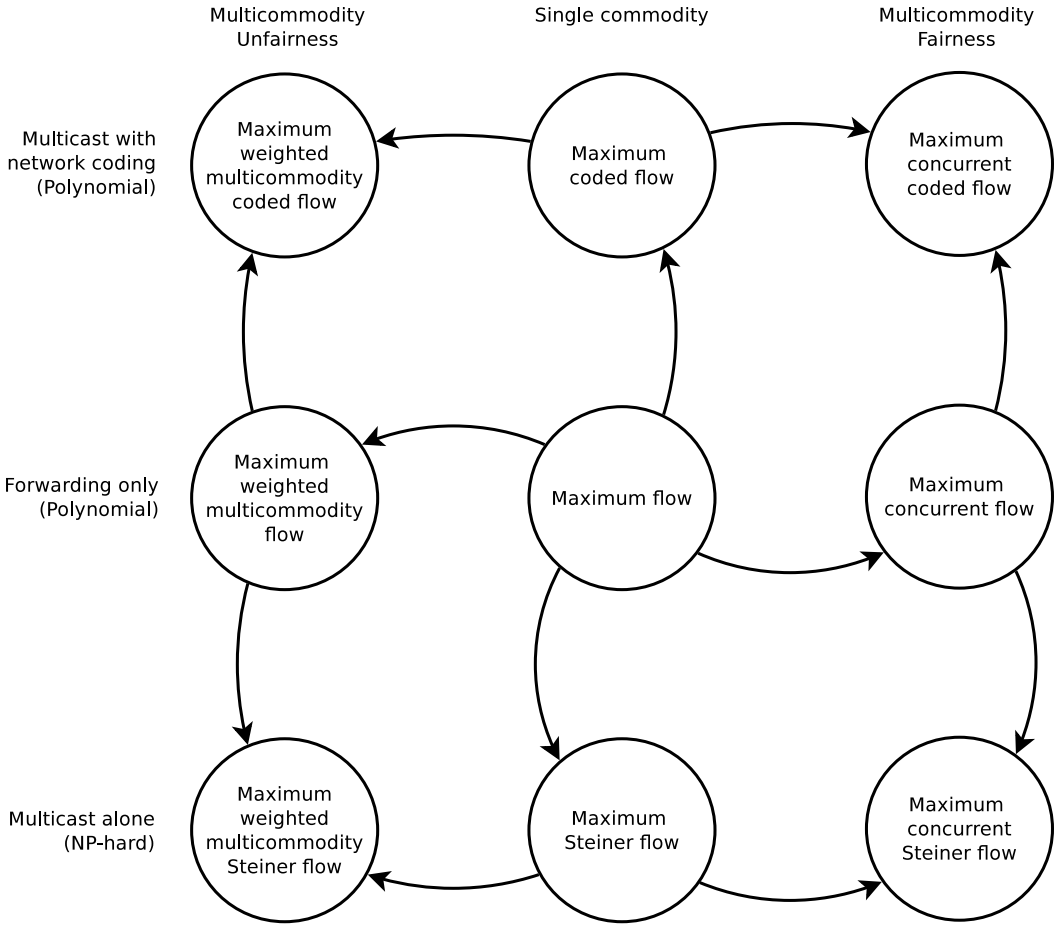


Figure 3.1 – Scheme of all problems studied in both the previous and the present chapters. An arrow from problem Π_A toward problem Π_B indicates that Π_B is a generalization of Π_A (or equivalently Π_A is a special case of Π_B). All problems on a same line share a common routing strategy (and a common algorithmic complexity) while all problems on a same column share a common feature.

commodity, indexed by an integer k, is a transportation request from a peculiar source node s^k to another peculiar receiver node r^k. Depending of the considered problem, a positive real *demand* d^k may also be specified for each commodity k. Let K be the set of all commodities.

For each commodity k, we denote by $\mathcal{P}^k$ the set of all simple paths from s^k to r^k. Let $\mathcal{P}$ be the union of the sets $\mathcal{P}^k$ over all commodities, $\mathcal{P} = \bigcup_{k \in K} \mathcal{P}^k$. Furthermore, for each arc a, and each commodity k, let $\mathcal{P}_a^k$ be the subset of $\mathcal{P}^k$ made of all paths using arc a. Finally, for each arc a, let $\mathcal{P}_a$ be the subset of $\mathcal{P}$ made of all paths using arc a. In other

words, $\mathcal{P}_a$ is the union of the sets $\mathcal{P}_a^k$ over all commodities, $\mathcal{P}_a = \bigcup_{k \in K} \mathcal{P}_a^k$.

A *multicommodity flow* x is an assignment of a non-negative real value x_p to each simple path p from s^k to r^k, for each commodity k. A multicommodity flow x is *feasible* if it satisfies the capacity requirements, namely that the sum over all commodities of the contribution of each path using an arc a is smaller than the capacity c_a of this arc. We denote by $\mathcal{X}_\mathcal{P}$ the set of all feasible multicommodity flows:

$$\mathcal{X}_\mathcal{P} = \left\{ x \in \mathbb{R}_+^\mathcal{P} : \sum_{p \in \mathcal{P}_a} x_p \leq c_a \; \forall a \in A \right\} \tag{3.1}$$

One can define a multicommodity flow problem by picking an objective function. We focus on maximizing a function of the throughput experienced by each receiver.

3.1.3.2 Maximum weighted multicommodity flow

Problem statement

According to the previous setting, consider the case where a non-negative weight w^k is associated to each commodity k. Given a multicommodity flow x, those weights are reflecting some preferences among commodities when the network operator aims to maximize the weighted sum of the throughput experienced by each receiver, referred to as the *value* of multicommodity flow x:

$$w(x) = \sum_{k \in K} w^k \sum_{p \in \mathcal{P}^k} x_p \tag{3.2}$$

where for each commodity k, $\sum_{p \in \mathcal{P}^k} x_p$ is the amount of flow going from source s^k to receiver r^k. This objective function leads to the so-called *maximum weighted multicommodity flow problem*:

Problem	*Maximum weighted multicommodity flow*
Instance	Network $D = (V, A)$, capacity c_a on each arc a, set of commodities K, and for each commodity k, source s^k, receiver r^k, non-negative weight w^k
Solution	A feasible multicommodity flow $x \in \mathcal{X}_\mathcal{P}$ in D
Objective	Maximize the value $w(x)$ of multicommodity flow x as defined in Equation (3.2)

We denote by ϕ_F the value of a maximum weighted multicommodity flow for the considered instance:

$$\phi_F = \max_{x \in \mathcal{X}_\mathcal{P}} w(x) \tag{3.3}$$

The special case where $w^k = 1$ for each commodity k is known as the *maximum multicommodity flow problem* in the literature. Notice that, when restricted to a single commodity ($|K| = 1$), the present problem amounts to finding a maximum flow with respect to this commodity.

Linear programming formulations and problem complexity

The maximum weighted multicommodity flow problem can be modelled using linear programming:

$$\begin{cases} \phi_F = \max \quad \displaystyle\sum_{k \in K} w^k \sum_{p \in \mathcal{P}^k} x_p & (3.4) \\ \text{s.t.} \quad \displaystyle\sum_{k \in K} \sum_{p \in \mathcal{P}_a^k} x_p \leq c_a \quad \forall a \in A & (3.5) \\ \qquad x_p \geq 0 \quad \forall p \in \mathcal{P} & (3.6) \end{cases}$$

where, for each simple path p, variable x_p stands for the amount of flow conveyed through p. Hence, this model may involve an exponential number of variables, due to the number of paths in $\mathcal{P}$. We will refer to the previous model as the *path formulation* of the maximum weighted multicommodity flow problem, so as to distinguish it from the *arc formulation* of the same problem. The aforementioned arc formulation leads to the following linear program:

$$\begin{cases} \phi_F = \max \quad \displaystyle\sum_{k \in K} w^k \phi^k & (3.7) \\ \text{s.t.} \quad \displaystyle\sum_{a \in \delta^+(v)} f_a^k - \sum_{a \in \delta^-(v)} f_a^k = b_v^k(\phi^k) \quad \forall v \in V, k \in K & (3.8) \\ \qquad \displaystyle\sum_{k \in K} f_a^k \leq c_a \quad \forall a \in A & (3.9) \\ \qquad \phi^k, f_a^k \geq 0 \quad \forall a \in A, k \in K & (3.10) \end{cases}$$

with

$$b_v^k(\phi) = \begin{cases} \phi & \text{if } v = s^k \quad (3.11) \\ -\phi & \text{if } v = r^k \quad (3.12) \\ 0 & \text{otherwise} \quad (3.13) \end{cases}$$

For each arc a and each commodity k, variable f_a^k stands for the amount of flow of commodity k going through a, while variable ϕ^k models the throughput experienced by receiver r^k.

The arc formulation of the maximum weighted multicommodity flow problem involves a polynomial number of variables and constraints (with respect to the instance size) so that the problem can be solved in polynomial time. Given an optimal solution f for the arc formulation, an optimal solution x for the path formulation can be built in polynomial time thanks to the technique presented in the proof of Lemma 1: For each commodity k, let f^k be the vector of components f_a^k over each arc a of the network. Observe that vector f^k is the arc projection of a *simple* flow between the source s^k and the receiver r^k. For each commodity k, compute a path decomposition of this simple flow using the algorithm presented by Ahuja et al. [7, Theorem 3.5] which runs in polynomial time. The concatenation x of all those path decompositions (one per commodity) is clearly a path decomposition of the multicommodity flow f. We refer the interested reader to the chapter devoted to multicommodity flows in [7] for a broader algorithmic perspective.

3.1.3.3 Maximum concurrent flow

The framework remains the same as previously. Consider the situation where a positive demand d^k is associated to each commodity k. The network operator wishes to satisfy as much of each commodity demand as possible, while ensuring strict fairness between the commodities. The objective is then to maximize the fraction of the demand experienced by every commodity, this last quantity being referred to as the *value* of a multicommodity flow x in what follows:

$$d(x) = \min_{k \in K} \frac{\sum_{p \in \mathcal{P}^k} x_p}{d^k} \tag{3.14}$$

where for each commodity k, $\sum_{p \in \mathcal{P}^k} x_p$ is the amount of flow going from source s^k to receiver r^k. This objective function leads to the so-called *maximum concurrent flow problem*:

Problem	*Maximum concurrent flow*
Instance	Network $D = (V, A)$, capacity c_a on each arc a, set of commodities K, and for each commodity k, source s^k, receiver r^k, positive demand d^k
Solution	A feasible multicommodity flow $x \in \mathcal{X}_{\mathcal{P}}$ in D
Objective	Maximize the value $d(x)$ of multicommodity flow x as defined in Equation (3.14)

We denote by λ_F the value of a maximum concurrent flow for the considered instance:

$$\lambda_F = \max_{x \in \mathcal{X}_{\mathcal{P}}} d(x) \tag{3.15}$$

Notice again that, when restricted to a single commodity ($|K| = 1$), the previous problem amounts to finding a maximum flow with respect to this commodity.

Linear programming formulations and problem complexity

The maximum concurrent flow problem can also be formalized through linear programming:

$$\begin{cases} \lambda_F = \max \quad \lambda & (3.16) \\ \text{s.t.} \quad d^k \lambda \leq \sum_{p \in \mathcal{P}^k} x_p \quad \forall k \in K & (3.17) \\ \qquad \sum_{k \in K} \sum_{p \in \mathcal{P}_a^k} x_p \leq c_a \quad \forall a \in A & (3.18) \\ \qquad x_p \geq 0 \quad \forall p \in \mathcal{P} & (3.19) \end{cases}$$

where, for each simple path p, variable x_p models the amount of flow conveyed through p. This model may also involve an exponential number of variables. The previous model is called the *path formulation* of the maximum concurrent flow problem, so as to distinguish it from its *arc formulation* whose linear program is given below:

$$\begin{cases} \lambda_F = \max \quad \lambda & (3.20) \\ \text{s.t.} \quad d^k \lambda \leq \phi^k \quad \forall k \in K & (3.21) \\ \qquad \sum_{a \in \delta^+(v)} f_a^k - \sum_{a \in \delta^-(v)} f_a^k = b_v^k(\phi^k) \quad \forall v \in V, k \in K & (3.22) \\ \qquad \sum_{k \in K} f_a^k \leq c_a \quad \forall a \in A & (3.23) \\ \qquad \phi^k, f_a^k \geq 0 \quad \forall a \in A, k \in K & (3.24) \end{cases}$$

where

$$b_v^k(\phi) = \begin{cases} \phi & \text{if } v = s^k \quad (3.25) \\ -\phi & \text{if } v = r^k \quad (3.26) \\ 0 & \text{otherwise} \quad (3.27) \end{cases}$$

As previously, for each arc a and each commodity k, variable f_a^k stands for the amount of flow of commodity k going through a, while variable ϕ^k models the throughput experienced by receiver r^k.

The arc formulation of the maximum concurrent flow problem involves a polynomial number of variables and constraints (with respect to the instance size) so that the problem can be solved in polynomial time. As before, given an optimal solution f for the arc formulation, an optimal solution x for the path formulation can be built in polynomial time thanks to the technique presented in the proof of Lemma 1.

3.1.3.4 Summary

We presented two examples of multicommodity flow problems which are well-known in the literature [7]. We will now study the extension of each problem to the framework of information flows.

3.2 Multicommodity Steiner flows

3.2.1 Setting

For the sake of simplicity, we focus our study on the case of a directed network, without loss of generality since it encompasses the case of an undirected one. The instance is made of a network $D = (V, A)$ with a capacity c_a for each arc a, and an index set K whose elements are called *commodities*, such that to each commodity k is associated a node s^k, the *source* for commodity k, and a subset R^k of terminal nodes, called *receivers* of commodity k. Depending on the considered application, a positive real *demand* d^k may also be specified for each commodity k. This setting allows one to model most cases of multiple multicast sessions, concurrently occurring in a given network, where each source independently conveys its own data to its receivers. We shall further assume that all

commodities are distinct, without loss of generality since we can add artificial sources and connect each of them to the network without altering the overall network structure. Notice however that we do *not* assume any two set of receivers, associated with two different commodities, to be distinct.

For each commodity k, we denote by $\mathcal{T}^k$ the set of all Steiner trees spanning s^k and R^k, and by $\mathcal{T}$ the union of the sets $\mathcal{T}^k$ over all commodities, $\mathcal{T} = \bigcup_{k \in K} \mathcal{T}^k$. Furthermore, for each arc a, and each commodity k, we denote by $\mathcal{T}_a^k$ the subset of $\mathcal{T}^k$ made of all trees using arc a. Finally, for each arc a, let $\mathcal{T}_a$ be the subset of $\mathcal{T}$ made of all trees using arc a. Equivalently $\mathcal{T}_a$ is the union of the sets $\mathcal{T}_a^k$ over all commodities, $\mathcal{T}_a = \bigcup_{k \in K} \mathcal{T}_a^k$.

A *multicommodity Steiner flow* x is an assignment of a non-negative real value x_t to each Steiner tree t spanning s^k and R^k, for each commodity k. A multicommodity Steiner flow x is *feasible* if it satisfies the capacity constraints, namely that the sum over all commodities of the contribution of each Steiner tree using an arc a is smaller than the capacity c_a of this arc. We denote by $\mathcal{X}_\mathcal{T}$ the set of all feasible multicommodity Steiner flows:

$$\mathcal{X}_\mathcal{T} = \left\{ x \in \mathbb{R}_+^\mathcal{T} : \sum_{t \in \mathcal{T}_a} x_t \leq c_a \, \forall a \in A \right\} \tag{3.28}$$

One may define a multicommodity Steiner flow problem by adding an objective function. We focus on the analog of each function previously used to define a multicommodity flow problem.

3.2.2 Maximum weighted multicommodity Steiner flow

3.2.2.1 Problem statement

Consider the framework where a non-negative *weight* w^k is associated to each commodity k. Practical examples of such weights are the *unit weight case*, $w_k = 1$ for each commodity k, or the *proportional weight case*, $w^k = \left(|R^k| / \sum_{k \in K} |R^k| \right)$. The aim is then to maximize the weighted sum, over all commodities, of the shared throughput experienced by each group of receivers within a commodity. Given a multicommodity Steiner flow x, we call *value* of x the quantity:

$$w(x) = \sum_{k \in K} w^k \sum_{t \in \mathcal{T}^k} x_t \tag{3.29}$$

where for each commodity k, $\sum_{t \in \mathcal{T}^k} x_t$ is the throughput provided by source s^k to each receiver in R^k. By a suitable choice of the weight vector w, the network operator can expect to achieve some balance in the capacity distribution between commodities. Observe however that a weighted sum may be a bad criterion if the network operator wishes to enforce strict fairness between the commodities. This objective function naturally leads to the following problem:

Problem	*Maximum weighted multicommodity Steiner flow*
Instance	Network $D = (V, A)$, capacity c_a on each arc a, set of commodities K, and for each commodity k, source s^k, set of receivers R^k, non-negative weight w^k
Solution	A feasible multicommodity Steiner flow $x \in \mathcal{X}_{\mathcal{T}}$ in D
Objective	Maximize the value $w(x)$ of multicommodity Steiner flow x as defined in Equation (3.29)

We denote by ϕ_S the value of a maximum weighted multicommodity Steiner flow for the considered instance:

$$\phi_S = \max_{x \in \mathcal{X}_{\mathcal{T}}} w(x) \tag{3.30}$$

Observe that, when restricted to a single commodity ($|K| = 1$), the present problem amounts to finding a maximum Steiner flow with respect to this commodity.

3.2.2.2 A detailed example

We consider the network depicted on Figure 3.2 which we shall refer to as the *directed bi-butterfly network*. This network is obtained by merging two copies of the directed butterfly network, while setting the capacity of each arc a of the network to 1, except for the arc (n_2, r_2) whose capacity is 2. We are looking for a maximum weighted multicommodity

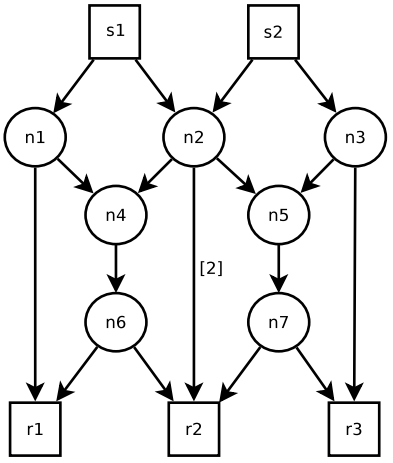


Figure 3.2 – The directed bi-butterfly network.

Steiner flow in the instance defined by setting two commodities in the directed bi-butterfly network with $s^1 = s_1$, $R^1 = \{r_1, r_2\}$, $s^2 = s_2$, and $R^2 = \{r_2, r_3\}$. Furthermore, the weight of each commodity is set to $\frac{1}{2}$. A maximum weighted multicommodity Steiner flow $\bowtie$ is depicted on Figure 3.3. Multicommodity Steiner flow $\bowtie$ uses three trees per commodity, each tree routing $\frac{1}{2}$ units of flow, so that the overall throughput induced by Steiner flow $\bowtie$ is $\frac{3}{2}$.

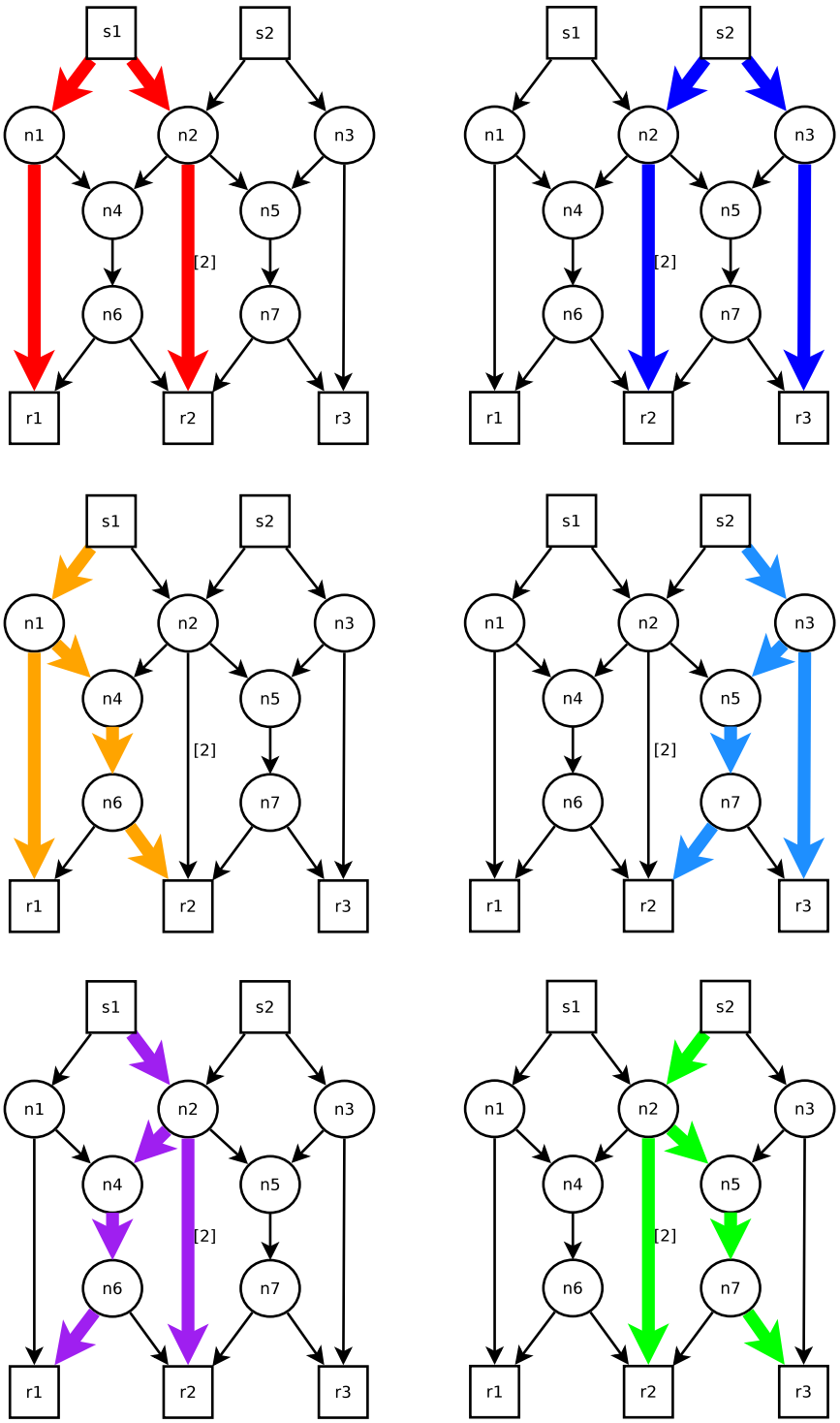


Figure 3.3 – A maximum multicommodity Steiner flow $\mathbb{x}$ using six trees in the directed bi-butterfly network. Each tree conveys 1/2 units of flow so that each receiver experiences a throughput of 3/2 *per commodity*. The overall throughput induced by Steiner flow $\mathbb{x}$ equals 3/2.

3.2.2.3 Linear programming formulation

The maximum weighted multicommodity Steiner flow problem can be formalized using linear programming. To the best of our knowledge, the formulation below was first proposed by Saad et al. [89] with the additional requirement to use one single Steiner tree per commodity:

$$\begin{cases} \phi_S = \max \quad \sum_{k \in K} w^k \sum_{t \in \mathcal{T}^k} x_t & (3.31) \\ \text{s.t.} \quad \sum_{k \in K} \sum_{t \in \mathcal{T}_a^k} x_t \leq c_a \quad \forall a \in A & (3.32) \\ \qquad x_t \geq 0 \quad \forall t \in \mathcal{T} & (3.33) \end{cases}$$

For each Steiner tree t, variable x_t stands for the amount of flow routed through t. Observe that, since we may have an exponential number of trees in $\mathcal{T}$, the previous model may also have an exponential number of variables. Constraints (3.32) ensure that the Steiner flow x is feasible. It is interesting to consider the dual of the above linear program:

$$\begin{cases} \phi_S = \min \quad \sum_{a \in A} c_a y_a & (3.34) \\ \text{s.t.} \quad \sum_{a \in t} y_a \geq w^k \quad \forall k \in K, t \in \mathcal{T}^k & (3.35) \\ \qquad y_a \geq 0 \quad \forall a \in A & (3.36) \end{cases}$$

where y is the vector of dual variables associated to Constraints (3.32). When $w^k = 1$ for each commodity k, the dual problem amounts to finding a multi-commodity Steiner multi-cut.

3.2.2.4 Complexity and algorithms

The separation problem associated to the dual of the above linear programming formulation of the maximum weighted multicommodity Steiner flow problem is to find a minimum-cost Steiner tree t with respect to arc cost function y. This last problem is NP-hard [43], hence by the equivalence between polynomial-time separation and polynomial-time optimization for convex polytopes [90], the dual problem is also NP-hard. Thus, the maximum weighted multicommodity Steiner flow problem is NP-hard.

To say more about this problem, we shall introduce the class of *fractional packing problems* to which the maximum weighted multicommodity Steiner flow problem belongs. We

shall then present a framework due to Garg and Könemann [21] to handle any problem of this class. Given a subroutine to solve the minimum Steiner tree problem with accuracy α, this framework provides an algorithm, running in polynomial time, to solve the maximum weighted multicommodity Steiner flow problem with ratio $\alpha(1+\epsilon)$, where ϵ is any fixed positive real number. This algorithm does not rely on the ellipsoid method and only incurs a factor $(1+\epsilon)$ loss in the approximation ratio. The reader only interested by the complexity result may skip the following discussion up to Corollary 2.

3.2.2.5 Approximation algorithm

Fractional packing, also known as *positive linear programming*, refers to a problem of the form $\max\{c^T x | Ax \leq b, x \geq 0\}$ where A is an $(m \times n)$ real matrix with non-negative entries, b is a positive vector in $\mathbb{R}^m$, and c is a positive vector in $\mathbb{R}^n$. Observe that the maximum weighted multicommodity Steiner flow problem is actually a packing problem. Let $I = [n]$ and $J = [m]$. Without loss of generality, every row and column of A contains at least one non-zero entry, and, for any couple of indices $(i,j) \in I \times J$, the $(i,j)^{th}$ entry of A, denoted by $A(i,j)$, is at most the i^{th} entry of b, denoted by b_i (see [91] for a discussion regarding those assumptions). Let A_j denote the j^{th} column of A for any $j \in J$. The dual of a given packing problem, called *the covering problem*, is the linear program $\min\{b^T y | A^T y \geq c, y \geq 0\}$. We further assume that the covering problem is feasible (the zero vector is always a feasible primal solution to the packing problem with value 0). By linear programming duality, the packing and covering problems share the same optimal value, and the value of any feasible solution of the covering problem is greater than, or equal to, the value of any feasible solution of the packing problem. The method proposed by Garg and Könemann in [21] allows to solve the packing problem provided an oracle which, given a dual vector y, computes *exactly* the quantity $\sigma(y) = \min_j\{A_j^T y / c_j\}$. Observe that computing $\sigma(y)$ amounts to solving the *separation problem* associated to the covering problem.

However, in many applications, we only have access to a *weak oracle* which, given a dual vector y, returns a column q such that $A_q^T y / c_q \leq \alpha\sigma(y)$ where α is the oracle *approximation ratio*. This ratio can be either a constant or a function of the size of the instance. When

$\alpha = 1$ the oracle is exact, but we may also have $\alpha = 1 + \epsilon$ for any desired accuracy ϵ (the oracle is then an FPTAS or a PTAS), or α equal to a constant, corresponding to the use of a constant approximation algorithm as oracle.

Saad et al. [89] show that the framework developed by Garg and Könemann [21] can be extended to cope with weak oracles. More precisely, the algorithm preserves the approximation ratio α from the oracle to the packing problem up to a $1 + \epsilon$ factor. The need for such an extension typically arises in some packing problems where there is an exponential number of possible columns (with respect to the size of the instance) implying that the matrix A and the vector c are only known through the oracle, while the dual separation problem is NP-hard but can be approximated to some extent.

Theorem 9. *[89] Given accuracy $\omega > 0$ and an oracle to find an α-approximate solution to the dual separation problem in time T_{Oracle}, the Garg and Könemann algorithm computes an $(1-\omega)/\alpha$-approximate solution to the packing problem, in time $O(\omega^{-2} m \log(m) T_{Oracle})$.*

When the oracle is an FPTAS or a PTAS, the previous result can be specialized as follows:

Corollary 1. *Given accuracy $\omega > 0$ and an FPTAS or a PTAS, which finds an $(1 + \epsilon)$-approximation to the dual separation problem in time T_{Oracle} (where $\epsilon(\omega)$ is chosen as above), the Garg and Könemann algorithm computes an $(1 - \omega)$-approximate solution to the packing problem, in time $O(\omega^{-3} m \log(m) T_{Oracle})$.*

The overall increase of the running time, from ω^{-2} to ω^{-3}, is due to the requirement for the quality of the approximation of the packing problem to be the same as the one of the oracle. For the sake of completeness we provide a full proof of Theorem 9, along with the pseudo-code of the associated algorithm, in Appendix B.1.

Assume we want to study a packing problem with an exponential number of columns so that the constraints matrix A is only known through an oracle. Observe that, in the definition of σ, the oracle approximately solve the *ratio variant* of the covering separation problem. If vector c is *known* to be equal to the all-ones vector then the ratio version is

equivalent to the *linear variant* of the covering separation problem from an *approximation perspective*. Otherwise, if the components of vector c are only provided through calls to the oracle, the very existence of such an oracle can be an issue since many "classical" approximation algorithms are designed to solve linear variants of hard problems. Nonetheless, it is sometimes possible to approximate the ratio variant given a subroutine to approximate the linear variant, see [92] and references therein.

As mentioned above, the separation problem associated to the dual of the maximum weighted multicommodity Steiner flow problem is to find a Steiner tree whose cost with respect to given dual vector y is minimum. Thus, one is looking for a solution to the following problem:

$$\min_{k \in K} \frac{1}{w^k} \min_{t \in \mathcal{T}^k} \sum_{a \in t} y_a \tag{3.37}$$

Hence, the following corollary follows:

Corollary 2. *Given accuracy $\omega > 0$, and an oracle to find an α-approximate minimum cost Steiner tree in time $T_{Steiner}$, the Garg and Könemann algorithm computes an $\alpha(1+\omega)$-approximate solution to the maximum weighted multicommodity Steiner flow problem, in time $O(\omega^{-2} m \log(m) T_{Steiner})$.*

We refer the reader to the discussion leading to Table 2.2 in the previous chapter, for a summary regarding the complexity and approximability of the minimum cost Steiner tree problem. Also see the paper by Ciebiera et al. [93] for a more practical perspective.

For our purpose, it is more meaningful to look for an exact method to solve the maximum weighted multicommodity Steiner flow problem, like column generation.

3.2.2.6 Computing a maximum weighted multicommodity Steiner flow by column generation

Although the problem of finding a Steiner tree of minimum cost is NP-hard [43], it can be solved to optimality by using one of its many classical mixed-integer linear programming formulations, see [55]. Hence, the separation problem in Equation (3.37) can be solved to optimality by performing one call to a mixed-integer linear programming solver for each commodity, so as to get one minimum-cost Steiner tree per commodity. By using this solver

as a subroutine in a column generation procedure, one can hope to compute a maximum weighted multicommodity Steiner flow in a reasonable amount of time, given an instance of reasonable size.

3.2.3 Maximum concurrent Steiner flow

3.2.3.1 Problem statement

As previously mentioned, a maximum weighted multicommodity Steiner flow may induce an unfair allocation of the network resources among commodities. Consider the framework where a positive *demand* d^k is associated to each commodity k. The network operator wishes to satisfy as much of each commodity demand as possible, while enforcing strict fairness between the commodities. The objective is then to maximize the fraction of the demand experienced by every commodity, this last quantity being referred to as the *value* of a multicommodity Steiner flow x:

$$d(x) = \min_{k \in K} \frac{\sum_{t \in \mathcal{T}^k} x_t}{d^k} \tag{3.38}$$

where for each commodity k, $\sum_{t \in \mathcal{T}^k} x_t$ is the throughput provided by source s^k to each receiver in R^k. This objective function leads to the following problem:

Problem	*Maximum concurrent Steiner flow*
Instance	Network $D = (V, A)$, capacity c_a on each arc a, set of commodities K, and, for each commodity k, source s^k, set of receivers R^k, and positive demand d^k
Solution	A feasible multicommodity Steiner flow $x \in \mathcal{X}_\mathcal{T}$ in D
Objective	Maximize the value $d(x)$ of multicommodity Steiner flow x as defined in Equation (3.38)

We denote by λ_S the value of a maximum concurrent Steiner flow for the considered instance:

$$\lambda_S = \max_{x \in \mathcal{X}_\mathcal{T}} d(x) \tag{3.39}$$

Observe that solving the restriction of the present problem to the single commodity setting ($|K| = 1$) amounts to finding a maximum Steiner flow with respect to this commodity.

3.2.3.2 Linear programming formulation

The maximum concurrent Steiner flow problem can be formalized using linear programming:

$$\begin{cases} \lambda_S = \max \quad \lambda & (3.40) \\ \text{s.t.} \quad d^k \lambda \leq \sum_{t \in \mathcal{T}^k} x_t \quad \forall k \in K & (3.41) \\ \quad \sum_{k \in K} \sum_{t \in \mathcal{T}_a^k} x_t \leq c_a \quad \forall a \in A & (3.42) \\ \quad x_t \geq 0 \quad \forall t \in \mathcal{T} & (3.43) \end{cases}$$

For each Steiner tree t, variable x_t stands for the amount of flow routed through t. Observe that, since we may have an exponential number of trees in $\mathcal{T}$, the previous model may have an exponential number of variables. The single variable λ models the satisfied fraction of each commodity demand. Constraints (3.41) along with the maximization of variable λ ensure that, for each commodity, as much of its demand as possible is satisfied, while Constraints (3.42) ensure that the Steiner flow x is feasible. It is also interesting to consider the dual of the above linear program:

$$\begin{cases} \lambda_S = \min \quad \sum_{a \in A} c_a y_a & (3.44) \\ \text{s.t.} \quad \sum_{k \in K} d^k z_k = 1 & (3.45) \\ \quad \sum_{a \in t} y_a \geq z_k \quad \forall k \in K, t \in \mathcal{T}^k & (3.46) \\ \quad y_a, z_k \geq 0 \quad \forall a \in A, k \in K & (3.47) \end{cases}$$

where y is the vector of dual variables associated to Constraints (3.42) while z is the one associated to Constraints (3.41). When $d^k = 1$ for each commodity k, the dual problem amounts to finding a set of Steiner multi-cuts.

3.2.3.3 Complexity and algorithms

Notice that the special case of the maximum concurrent Steiner flow problem where there is only one single commodity with unit demand is exactly the problem of packing Steiner trees fractionally so as to maximize the overall throughput experienced by each receiver. Since this special case is already NP-hard [40], so is the maximum concurrent Steiner flow problem.

In the following, we shall introduce the class of *max-min resource sharing problems* which includes the maximum concurrent Steiner flow problem. We shall then present a framework also due to Garg and Könemann [21] and later extended by Fleischer [94, 22] to handle any problem of this class. Given a subroutine to solve the minimum Steiner tree problem with accuracy α, this framework provides an algorithm, running in polynomial time, to solve the maximum concurrent Steiner flow problem within a ratio $\alpha(1+\epsilon)$, where ϵ is any fixed positive real number. We stress out that it is not possible to simply apply the argument provided by Jain et al. in [40] to reach a similar conclusion through the ellipsoid method, because of the shape of the dual objective function. Again, the reader only interested by the complexity result may skip the following discussion up to Corollary 3.

3.2.3.4 Approximation algorithm

The *linear max-min resource sharing problem* can be thought of as a generalisation of the packing problem. It is of the form $\max\{\lambda | d\lambda \leq Hx, Ax \leq b, x \geq 0\}$ where A is an $(m \times n)$ real matrix with non-negative entries, b is a positive vector in $\mathbb{R}^m$, d is a positive vector in $\mathbb{R}^h$, and H is an $(h \times n)$ block diagonal binary matrix, such that each block is made of a single row where every coefficient is a 1. Observe that the maximum concurrent Steiner flow problem is actually a linear max-min resource sharing problem. Let K be a set indexing the rows of H, whose elements are called *commodities*. We assume, without loss of generality, that each such row k has at least one non-zero entry and we denote by C^k the set of all columns j of matrix H such that $H(k,j) = 1$. The max-min resource sharing problem was previously studied by Fleischer and Wayne in [22] under the name of *the k-commodity packing problem*. Its dual is $\min\{b^T y | d^T z = 1, A^T y \geq H^T z, y, z \geq 0\}$. We assume that the dual problem is feasible (the zero vector is always a feasible primal solution to the max-min resource sharing problem with value 0). In [22], the authors use the Garg-Könemann method to design an FPTAS given an *exact* oracle to solve a variant of the dual separation problem. However, we may only have access to a weak oracle for solving the dual separation problem. As already pointed out in [22], when $h = 1$, the max-min resource sharing problem becomes equivalent to the packing problem.

It is possible to adapt the framework of Fleischer and Wayne [22], which is based on the

method developed by Garg and Könemann in [21] for solving the maximum concurrent flow problem, to cope with a *weak oracle*. Again, the algorithm preserves the approximation ratio α from the oracle to the max-min resource sharing problem up to a $1+\epsilon$ factor.

Theorem 10. *Given accuracy $\omega > 0$ and an oracle to find an α-approximate solution to the dual separation problem in time T_{Oracle}, the Garg, Könemann, Fleischer, and Wayne algorithm computes an $(1-\omega)/\alpha$-approximate solution to the max-min resource sharing problem, in time $O(\omega^{-2}(m + h\log(h))\log(m)T_{Oracle})$.*

A full proof of Theorem 10, along with the pseudo-code of the associated algorithm, can be found in Appendix B.2.

The separation problem associated to the dual of the maximum concurrent Steiner flow problem is to find, for each commodity k, a Steiner tree rooted at s^k, spanning R^k, and whose cost with respect to given dual vector y is minimum. This implies the following corollary:

Corollary 3. *Given accuracy $\omega > 0$ and an oracle to find an α-approximate minimum cost Steiner tree in time $T_{Steiner}$, the Garg-Könemann-Fleischer-Wayne algorithm computes an $\alpha(1+\omega)$-approximate solution to the maximum concurrent Steiner flow problem, in time $O(\omega^{-2}(m + |K|\log(|K|))\log(m)T_{Steiner})$.*

We again refer the reader to the discussion leading to Table 2.2 in the previous chapter, for a summary regarding the complexity and approximability of the minimum cost Steiner tree problem. Also see the paper by Ciebiera et al. [93] for implementations of some algorithms. To conclude the present discussion, we would like to point out that an approach similar to the one presented here, combining the framework of Garg and Könemann with an approximation algorithm as subroutine, has been provided by Baltz et al. in [95] to solve the twin problem of finding a maximum concurrent Steiner flow, namely the *minimum multicast congestion problem*. See also the paper by Jansen et al. for more on this last problem [96].

For our purpose, it is more interesting to look for an exact method to solve the maximum concurrent Steiner flow problem, like column generation.

3.2.3.5 Computing a maximum concurrent Steiner flow by column generation

The separation problem associated to the dual of the maximum concurrent Steiner flow problem can be solved to optimality by performing one call to a mixed-integer linear programming solver for each commodity. By using this solver as a subroutine in a column generation procedure, one can hope to compute a maximum concurrent Steiner flow in a reasonable amount of time, given an instance of reasonable size.

We shall now focus on the setting where coding techniques are allowed in the network.

3.3 Multicommodity coded flows

3.3.1 Setting

The setting is the same as before. As in the previous chapter, we shall distinguish between directed and bidirected networks on the one hand, and undirected networks on the other hand. To summarize, the undirected network model is again obtained by adding *orientation constraints* to the bidirected network model. The instance is still made of a network $D = (V, A)$ (directed or bidirected, for the sake of simplicity) with a capacity c_a for each arc a, along with an index set K whose elements are called *commodities*, such that to each commodity k is associated a *source* s^k and a subset R^k of *receivers*. We may also be given a positive real *demand* d^k for each commodity k. We shall again assume that all commodities are distinct (without loss of generality, as previously explained). Also recall that we do *not* assume any two sets of receivers, associated with two different commodities, to be distinct. That said, we allow *intra-commodity coding*, but not *inter-commodity coding*: it is only possible to code messages originating from the same source. This prevents potential security leaks where a receiver from one commodity could decode information from another one. Our results hence differ from other approaches like the one undertaken by Cassuto and Bruck [25] and further extended by Chekuri et al. [27], where one wishes to use coding techniques so as to route data inside a network, from a single source toward a set of receivers, each receiver having its own demand.

For each commodity k, and each receiver $r \in R^k$, we denote by $\mathcal{P}^{(k,r)}$ the set of all simple

paths between s^k and r, and by $\mathcal{P}$ the union of the sets $\mathcal{P}^{(k,r)}$ over all commodities and all receivers, $\mathcal{P} = \bigcup_{k\in K} \bigcup_{r\in R^k} \mathcal{P}^{(k,r)}$. Furthermore, for each arc a, each commodity k, and each receiver $r \in R^k$, let $\mathcal{P}_a^{(k,r)}$ be the subset of $\mathcal{P}^{(k,r)}$ made of all paths using arc a. Finally, for each arc a, let $\mathcal{P}_a$ be the subset of $\mathcal{P}$ made of all paths using a. Equivalently $\mathcal{P}_a$ is the union of the sets $\mathcal{P}_a^{(k,r)}$ over all commodities and all receivers, $\mathcal{P}_a = \bigcup_{k\in K} \bigcup_{r\in R^k} \mathcal{P}_a^{(k,r)}$.

A *multicommodity coded flow* x is an assignment of a non-negative real value x_p to each simple path p between a source s^k and a receiver $r \in R^k$, for each commodity k. The multicommodity coded flow x is *feasible* if it satisfies the capacity constraints, namely that the sum over all commodities of the contribution of each *coded flow* using an arc a is smaller than the capacity c_a of this arc. This means that given a multicommodity coded flow x, along with an arc a, the amount of coded flow going through a for commodity k *alone* is given by:

$$h_a^k = \max_{r\in R^k} \sum_{p\in\mathcal{P}_a^{(k,r)}} x_p \tag{3.48}$$

while the overall used capacity is taken as the sum of those terms over all commodities, namely:

$$\hbar_a = \sum_{k\in K} h_a^k \tag{3.49}$$

Those equations directly translate our allowance of *intra-commodity coding* along with the prohibition of *inter-commodity coding*. In the following, we denote by $\mathcal{X}_\mathcal{P}$ the set of all feasible multicommodity coded flows

$$\mathcal{X}_\mathcal{P} = \left\{ x \in \mathbb{R}_+^\mathcal{P} : \sum_{k\in K} \max_{r\in R^k} \sum_{p\in\mathcal{P}_a^{(k,r)}} x_p \leq c_a \ \forall a \in A \right\} \tag{3.50}$$

Again, one may define a multicommodity coded flow problem by adding an objective function. In the following, we consider the two functions previously studied in this chapter.

3.3.2 Maximum weighted multicommodity coded flow

3.3.2.1 Problem statement

Consider again the framework where a non-negative *weight* w^k is associated to each commodity k. The aim is still to maximize the weighted sum, over all commodities, of the

shared throughput experienced by each group of receivers within a commodity. Given a multicommodity coded flow x, for each commodity k, the minimum throughput experienced by each receiver in R^k is:

$$\phi^k = \min_{r \in R^k} \sum_{p \in \mathcal{P}^{(k,r)}} x_p \tag{3.51}$$

(where, for each commodity k, $\sum_{p \in \mathcal{P}^{(k,r)}} x_p$ is the throughput provided by source s^k to receiver $r \in R^k$) so that the *value* of multicommodity coded flow x is given by:

$$w(x) = \sum_{k \in K} w^k \phi^k \tag{3.52}$$

This objective function gives rise to the following problem:

Problem	*Maximum weighted multicommodity coded flow*
Instance	Network $D = (V, A)$, capacity c_a on each arc a, set of commodities K, and, for each commodity k, source s^k, set of receivers R^k, non-negative weight w^k
Solution	A feasible multicommodity coded flow $x \in \mathcal{X}_{\mathcal{P}}$ in D
Objective	Maximize the value $w(x)$ of multicommodity coded flow x as defined in Equation (3.52)

We denote by ϕ_C the value of a maximum weighted multicommodity coded flow for the considered instance:

$$\phi_C = \max_{x \in \mathcal{X}_{\mathcal{P}}} w(x) \tag{3.53}$$

Observe that solving the restriction of the present problem to the single commodity setting ($|K| = 1$) amounts to finding a maximum coded flow with respect to this commodity.

3.3.2.2 Example continued

Consider again the directed bi-butterfly network depicted on Figure 3.2. This time, we are looking for a maximum multicommodity coded flow in the directed bi-butterfly network under the same setting as in the previous example, namely $s^1 = s_1$, $R^1 = \{r_1, r_2\}$, $s^2 = s_2$, and $R^2 = \{r_2, r_3\}$, whereas the weight of each commodity is set to $\frac{1}{2}$. A coding scheme associated to a maximum weighted multicommodity coded flow $\varkappa$ for this instance is depicted on Figure 3.4. The overall throughput induced by coded flow $\varkappa$ is 2 since each receiver experiences a throughput of 2 units of flow *per commodity*.

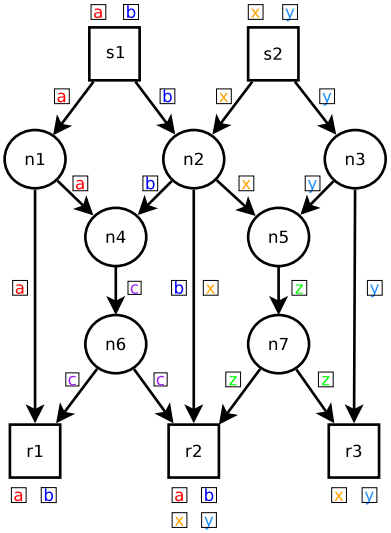


Figure 3.4 – A coding scheme associated to a maximum weighted multicommodity coded flow $\mathcal{X}$ in the directed bi-butterfly network. Each receiver experiences a throughput of 2 units of flow *per commodity*.

3.3.2.3 Linear programming formulations

Directed and bidirected networks

In a directed or bidirected network, the maximum weighted multicommodity coded flow problem can be formalized using linear programming:

$$\begin{cases} \phi_C = \max \quad \displaystyle\sum_{k \in K} w^k \phi^k & (3.54) \\ \text{s.t.} \quad \phi^k = \displaystyle\sum_{p \in \mathcal{P}^{(k,r)}} x_p \quad \forall k \in K, r \in R^k & (3.55) \\ \quad \displaystyle\sum_{p \in \mathcal{P}_a^{(k,r)}} x_p \leq h_a^k \quad \forall a \in A, k \in K, r \in R^k & (3.56) \\ \quad \displaystyle\sum_{k \in K} h_a^k \leq c_a \quad \forall a \in A & (3.57) \\ \quad h_a^k, x_p \geq 0 \quad \forall a \in Az, k \in K, p \in \mathcal{P} & (3.58) \end{cases}$$

For each simple path p, variable x_p stands for the amount of flow routed along p. Observe that, since we may have an exponential number of paths in $\mathcal{P}$, the previous model may also have an exponential number of variables. For each arc a and each commodity k, variable h_a^k models the amount of coded flow going through a for commodity k. Notice that each variable h_a^k satisfies Equation (3.48) in an optimal solution. Furthermore, for

each commodity k, variable ϕ^k stands for the minimum amount of flow experienced by any receiver in R^k, thanks to Constraints (3.55) along with the shape of the objective function, so that each variable ϕ^k satisfies Equation (3.51) in an optimal solution. Finally, Constraints (3.56)-(3.57) ensure that the coded flow x is feasible. Beware that the above *path formulation* of the studied problem can *not* be further decomposed into a set of easier sub-problems since the commodities are competing for bandwidth access.

The following linear program can be used to solve the *arc formulation* of the maximum weighted multicommodity coded flow problem (in a directed or bidirected network):

$$\begin{cases} \max \quad \sum_{k \in K} w^k \phi^k & (3.59) \\ \text{s.t.} \quad \sum_{a \in \delta^+(v)} f_a^{(k,r)} - \sum_{a \in \delta^-(v)} f_a^{(k,r)} = b_v^{(k,r)}(\phi^k) \quad \forall k \in K, r \in R^k, v \in V & (3.60) \\ \quad f_a^{(k,r)} \leq h_a^k \quad \forall a \in A, k \in K, r \in R^k & (3.61) \\ \quad \sum_{k \in K} h_a^k \leq c_a \quad \forall a \in A & (3.62) \\ \quad f_a^{(k,r)}, h_a^k \geq 0 \quad \forall a \in A, k \in K & (3.63) \end{cases}$$

where

$$b_v^{(k,r)}(\phi) = \begin{cases} \phi & \text{if } v = s^k \quad (3.64) \\ -\phi & \text{if } v = r^k \quad (3.65) \\ 0 & \text{otherwise} \quad (3.66) \end{cases}$$

For each arc a, each commodity k, and each receiver $r \in R^k$, variable $f_a^{(k,r)}$ stands for the amount of flow of commodity k, going from source s^k to receiver r through arc a.

Undirected networks

The linear program below, inspired by the work of Li et al. [60, 39], allows to *define* and *compute* a maximum weighted multicommodity coded flow in any undirected network. An instance of the considered problem is made of an undirected network $G = (V, E)$ with capacity c_e for each edge e, and a set K of commodities with, for each commodity k, a source s^k, a set of receivers R^k, and a non-negative weight w^k. We shall now use the same idea as when defining a maximum coded flow in an undirected network: Consider the bidirected network $B = (V, L)$ obtained by replacing each edge of G by a *pair of reverse*

arcs. Keep the sources and the receivers as defined in G. Finally, assign a *variable capacity* $\hbar_a$ on each arc a of the bidirected network then add an *orientation constraint* for each link ℓ. The following linear program allows to look for a maximum weighted multicommodity coded flow over all possible orientations of the undirected network:

$$\begin{cases} \phi_C = \max \quad \displaystyle\sum_{k \in K} w^k \phi^k & (3.67) \\ \text{s.t.} \quad \phi^k = \displaystyle\sum_{p \in \mathcal{P}^{(k,r)}} x_p \quad \forall k \in K, r \in R^k & (3.68) \\ \quad \displaystyle\sum_{p \in \mathcal{P}_a^{(k,r)}} x_p \leq h_a^k \quad \forall a \in A, k \in K, r \in R^k & (3.69) \\ \quad \displaystyle\sum_{k \in K} h_a^k \leq \hbar_a \quad \forall a \in A, k \in K & (3.70) \\ \quad \hbar_{a_1} + \hbar_{a_2} \leq c_\ell \quad \forall \ell = \{a_1, a_2\} \in L & (3.71) \\ \quad h_a^k, \hbar_a, x_p \geq 0 \quad \forall a \in A, k \in K, p \in \mathcal{P} & (3.72) \end{cases}$$

For each simple path p, variable x_p still stands for the amount of flow routed along p. (The proposed model may have an exponential number of variables.) For each arc a and each commodity k, variable h_a^k models the amount of coded flow going through a for commodity k, while variable $\hbar_a$ accounts for the variable arc capacity. Variable ϕ^k stands for the minimum amount of flow experienced by any receiver in R^k, thanks to Constraints (3.68) along with the shape of the objective function. Constraints (3.69) and Constraints (3.70) can be respectively regarded as intra-commodity and inter-commodity capacity requirements, with a *variable capacity* $\hbar_a$ for each arc a. Finally, Constraints (3.71) are the previously mentioned *orientation constraints*.

The previous linear program allows to solve the *path formulation* of the studied problem. It is also interesting to consider the linear program below which provides an *arc formulation*

of the same problem:

$$\begin{cases} \max \quad \sum_{k \in K} w^k \phi^k & (3.73) \\ \text{s.t.} \quad \sum_{a \in \delta^+(v)} f_a^{(k,r)} - \sum_{a \in \delta^-(v)} f_a^{(k,r)} = b_v^{(k,r)}(\phi^k) \quad \forall k \in K, r \in R^k, v \in V & (3.74) \\ \quad f_a^{(k,r)} \leq h_a^k \quad \forall a \in A, k \in K, r \in R^k & (3.75) \\ \quad \sum_{k \in K} h_a^k \leq \hbar_a \quad \forall a \in A, k \in K & (3.76) \\ \quad \hbar_{a_1} + \hbar_{a_2} \leq c_\ell \quad \forall \ell = \{a_1, a_2\} \in L & (3.77) \\ \quad f_a^{(k,r)}, h_a^k, \hbar_a \geq 0 \quad \forall a \in A, k \in K, r \in R^k & (3.78) \end{cases}$$

with

$$b_v^{(k,r)}(\phi) = \begin{cases} \phi & \text{if } v = s^k \quad (3.79) \\ -\phi & \text{if } v = r^k \quad (3.80) \\ 0 & \text{otherwise} \quad (3.81) \end{cases}$$

where for each arc a, each commodity k, and each receiver r, variable $f_a^{(k,r)}$ stands for the amount of flow from source s^k to receiver r going through arc a, variable h_a^k models the amount of coded flow going through a for commodity k, and variable $\hbar_a$ is the variable arc capacity. Constraints (3.74) enforce flow conservation at each vertex, while the role played by the other constraints is the same as in the path formulation.

3.3.2.4 Complexity and algorithms

Regardless of the network structure, the linear program obtained by considering the arc formulation of the maximum weighted coded flow problem involves a polynomial number of continuous variables along with a polynomial number of constraints (with respect to the instance size). Hence, it is possible to get the arc decomposition of a maximum weighted multicommodity coded flow in polynomial time. Given such an arc decomposition f, a path decomposition x can be built in polynomial time thanks to the technique presented in the proof of Lemma 1: For each commodity k, and each receiver $r \in R^k$, let $f^{(k,r)}$ be the vector of components $f_a^{(k,r)}$ over each arc a of the network. Observe that vector $f^{(k,r)}$ is the arc projection of a *simple* flow between source s^k and receiver r. For each commodity k and each receiver $r \in R^k$, compute a path decomposition of this simple flow using the

algorithm presented by Ahuja et al. [7, Theorem 3.5]. The concatenation x of all those path decompositions (one per receiver of each commodity) is clearly a path decomposition of the multicommodity coded flow f. Observe that this strategy involves the computation of $\sum_{k\in K} |R^k|$ path decompositions of a simple flow, which can be done in polynomial time.

3.3.2.5 Coding scheme

So far, we focused on the routing aspect of the multicommodity coded flow. Now, we turn our attention to the coding scheme design. Given a maximum weighted multicommodity coded flow x along with its associated vector f, one can use any classical network coding design algorithm to get a coding scheme for each commodity, independently from the other commodities: For each commodity k, give to the aforementioned coding scheme design algorithm the network G with capacity h_a^k on each arc a as input. This technique is referred to as *coding by superposition* in the literature, see [60, 39, 15] and references therein.

3.3.3 Maximum concurrent coded flow

3.3.3.1 Problem statement

Consider again the framework where a positive *demand* d^k is associated to each commodity k. Recall that given a multicommodity coded flow x, for each commodity k, the minimum throughput experienced by each receiver in R^k is:

$$\phi^k = \min_{r\in R^k} \sum_{p\in\mathcal{P}^{(k,r)}} x_p \tag{3.82}$$

The objective is still to maximize the fraction of the demand experienced by every commodity, this last quantity being referred to as the *value* of a multicommodity coded flow x:

$$d(x) = \min_{k\in K} \frac{\phi^k}{d^k} \tag{3.83}$$

This objective function leads to the following problem:

Problem	*Maximum concurrent coded flow*
Instance	Network $D = (V, A)$, capacity c_a on each arc a, set of commodities K, and, for each commodity k, source s^k, set of receivers R^k, positive demand d^k
Solution	A feasible multicommodity coded flow $x \in \mathcal{X}_\mathcal{P}$ in D
Objective	Maximize the value $d(x)$ of multicommodity coded flow x as defined in Equation (3.83)

In the following, we denote by λ_C the value of a maximum concurrent coded flow for the considered instance:

$$\lambda_C = \max_{x \in \mathcal{X}_\mathcal{P}} d(x) \tag{3.84}$$

Observe that, when restricted to a single commodity ($|K| = 1$), the present problem amounts to finding a maximum coded flow with respect to this commodity.

3.3.3.2 Linear programming formulations

Directed and bidirected networks

In a directed or bidirected network, the maximum concurrent coded flow problem can be formulated through linear programming as follows:

$$\begin{cases} \lambda_C = \max \quad \lambda & (3.85) \\ \text{s.t.} \quad d^k \lambda \leq \sum_{p \in \mathcal{P}^{(k,r)}} x_p \quad \forall k \in K, r \in R^k & (3.86) \\ \quad \sum_{p \in \mathcal{P}_a^{(k,r)}} x_p \leq h_a^k \quad \forall a \in A, k \in K, r \in R^k & (3.87) \\ \quad \sum_{k \in K} h_a^k \leq c_a \quad \forall a \in A & (3.88) \\ \quad h_a^k, x_p \geq 0 \quad \forall a \in A, k \in K, p \in \mathcal{P} & (3.89) \end{cases}$$

For each simple path p, variable x_p stands for the amount of flow routed along p. We may again end up dealing with an exponential number of such variables. For each arc a and each commodity k, variable h_a^k models the amount of coded flow going through a for commodity k. Observe that each variable h_a^k satisfies Equation (3.48) in an optimal solution. Variable λ stands for the minimum fraction of the demand experienced by any receiver, thanks to Constraints (3.86) along with the shape of the objective function. Finally, Constraints (3.87)-(3.88) ensure that the multicommodity coded flow x satisfies all capacity requirements. Again, the above *path formulation* of the present problem can *not* be fur-

ther decomposed into a set of easier sub-problems due to the competition for bandwidth between the commodities.

The linear program below can be used to solve the *arc formulation* of the maximum concurrent coded flow problem (in a directed or bidirected network):

$$\begin{cases} \max \quad \lambda & (3.90) \\ \text{s.t.} \quad d^k \lambda \leq \phi^k \quad \forall k \in K & (3.91) \\ \quad \sum_{a \in \delta^+(v)} f_a^{(k,r)} - \sum_{a \in \delta^-(v)} f_a^{(k,r)} = b_v^{(k,r)}(\phi^k) \quad \forall k \in K, r \in R^k, v \in V & (3.92) \\ \quad f_a^{(k,r)} \leq h_a^k \quad \forall a \in A, k \in K, r \in R^k & (3.93) \\ \quad \sum_{k \in K} h_a^k \leq c_a \quad \forall a \in A & (3.94) \\ \quad f_a^{(k,r)}, h_a^k \geq 0 \quad \forall a \in A, k \in K & (3.95) \end{cases}$$

where

$$b_v^{(k,r)}(\phi) = \begin{cases} \phi & \text{if } v = s^k \quad (3.96) \\ -\phi & \text{if } v = r^k \quad (3.97) \\ 0 & \text{otherwise} \quad (3.98) \end{cases}$$

For each arc a, each commodity k, and each receiver $r \in R^k$, variable $f_a^{(k,r)}$ stands for the amount of flow of commodity k, going from source s^k to receiver r through arc a.

Undirected networks

Li et al. [60, 39] proposed the linear program below so as to *define* and *compute* a maximum concurrent coded flow in an undirected network. An instance of the problem is made of an undirected network $G = (V, E)$ with capacity c_e for each edge e, and a set K of commodities with, for each commodity k, a source s^k, a set of receivers R^k, and a positive demand d^k. We shall use the same trick as previously by introducing the bidirected network $B = (V, L)$ naturally associated to G, with a *variable capacity* h_a on each arc a of the bidirected network along with an *orientation constraint* for each link ℓ. The next linear program allows to find a maximum concurrent coded flow over all possible orientations of

the undirected network:

$$\begin{cases} \lambda_C = \max \quad \lambda & (3.99) \\ \text{s.t.} \quad d^k \lambda \leq \sum_{p \in \mathcal{P}^{(k,r)}} x_p \quad \forall k \in K, r \in R^k & (3.100) \\ \quad \sum_{p \in \mathcal{P}_a^{(k,r)}} x_p \leq h_a^k \quad \forall a \in A, k \in K, r \in R^k & (3.101) \\ \quad \sum_{k \in K} h_a^k \leq \hbar_a \quad \forall a \in A, k \in K & (3.102) \\ \quad \hbar_{a_1} + \hbar_{a_2} \leq c_\ell \quad \forall \ell = \{a_1, a_2\} \in L & (3.103) \\ \quad h_a^k, \hbar_a, x_p \geq 0 \quad \forall a \in A, k \in K, p \in \mathcal{P} & (3.104) \end{cases}$$

For each simple path p, variable x_p still stands for the amount of flow routed along p. (The model may have an exponential number of variables.) For each arc a and each commodity k, variable h_a^k models the amount of coded flow going through a for commodity k, while variable $\hbar_a$ is the variable arc capacity. Variable λ stands for the minimum fraction of the demand experienced by any receiver, thanks to Constraints (3.100) along with the shape of the objective function. Constraints (3.101) and Constraints (3.102) are respectively intra-commodity and inter-commodity capacity requirements, with a *variable capacity* $\hbar_a$ for each arc a. Finally, Constraints (3.103) are the previously mentioned *orientation constraints*.

The previous linear program allows to solve the *path formulation* of the studied problem. As before, we also consider the linear program below which is associated to the *arc formulation* of the same problem:

$$\begin{cases} \lambda_C = \max \quad \lambda & (3.105) \\ \text{s.t.} \quad d^k \lambda \leq \phi^k \quad \forall k \in K, r \in R^k & (3.106) \\ \quad \sum_{a \in \delta^+(v)} f_a^{(k,r)} - \sum_{a \in \delta^-(v)} f_a^{(k,r)} = b_v^{(k,r)}(\phi^k) \quad \forall k \in K, r \in R^k, v \in V & (3.107) \\ \quad f_a^{(k,r)} \leq h_a^k \quad \forall a \in A, k \in K, r \in R^k & (3.108) \\ \quad \sum_{k \in K} h_a^k \leq \hbar_a \quad \forall a \in A, k \in K & (3.109) \\ \quad \hbar_{a_1} + \hbar_{a_2} \leq c_\ell \quad \forall \ell = \{a_1, a_2\} \in L & (3.110) \\ \quad f_a^{(k,r)}, h_a^k, \hbar_a \geq 0 \quad \forall a \in A, k \in K, r \in R^k & (3.111) \end{cases}$$

with

$$b_v^{(k,r)}(\phi) = \begin{cases} \phi & \text{if } v = s^k \quad (3.112) \\ -\phi & \text{if } v = r^k \quad (3.113) \\ 0 & \text{otherwise} \quad (3.114) \end{cases}$$

where for each arc a, each commodity k, and each receiver r, variable $f_a^{(k,r)}$ stands for the amount of flow from source s^k to receiver r going through arc a, variable h_a^k models the amount of coded flow going through a for commodity k, and variable $\hbar_a$ is the arc capacity. Constraints (3.107) enforce flow conservation at each vertex, while the role played by the other constraints is the same as in the path formulation.

3.3.3.3 Complexity and algorithms

Regardless of the network structure, the arc formulation of the maximum concurrent coded flow problem involves a polynomial number of (continuous) variables and constraints (with respect to the instance size) so that an arc decomposition of a maximum concurrent coded flow can be computed in polynomial time. In order to get an equivalent path decomposition of this multicommodity coded flow in polynomial time, simply use the technique previously described in the proof of Lemma 1, which involves the computation of $\sum_{k \in K} |R^k|$ path decompositions of a simple flow in the present setting (see the discussion above regarding the complexity of the maximum weighted multicommodity coded flow problem).

3.3.3.4 Coding scheme

Again, given a maximum concurrent coded flow x along with its associated vector h, one can use any classical network coding design algorithm to get a coding scheme for each commodity, independently from the other commodities: for each commodity k, give to the coding scheme design algorithm the network D with capacity h_a^k on each arc a as input. We refer the interested reader to [60, 39, 15] for more details regarding *coding by superposition*.

3.4 Features of the information flows

3.4.1 Multicommodity coding gains

Considering the results presented in the first chapter regarding the coding gain, it seems quite natural to study the benefits of using coding techniques in the present framework.

Question 2. *Is there an incentive, with respect to the maximum weighted or concurrent throughput, to use network coding in the multiple-multicast setting ?*

To answer this question one needs again a proper criterion to evaluate the benefit of deploying coding techniques inside the network. The following two quantities seem quite natural.

Definition 2. *Given a network $D = (V, A)$, set of commodities K with for each commodity k, source s^k, set of receivers R^k, and non-negative weight w^k, along with capacity c_a for each arc a, the* multicommodity coding gain *g_ϕ, provided by network coding over multicast alone, is the ratio of the value of a maximum weighted multicommodity coded flow over the one of a maximum weighted multicommodity Steiner flow, namely:*

$$g_\phi = \frac{\phi_C}{\phi_S} \tag{3.115}$$

Definition 3. *Given a network $D = (V, A)$, set of commodities K with for each commodity k, source s^k, set of receivers R^k, and positive demand d^k, along with capacity c_a for each arc a, the* concurrent coding gain *g_λ, provided by network coding over multicast alone, is the ratio of the value of a maximum concurrent coded flow over the one of a maximum concurrent Steiner flow, namely:*

$$g_\lambda = \frac{\lambda_C}{\lambda_S} \tag{3.116}$$

Since we restrict our attention to networks where, for each commodity k, there is at least one path between source s^k and each receiver $r \in R^k$, it is safe to assume both $\phi_S > 0$ and $\lambda_S > 0$, since there exists at least one Steiner tree per commodity to convey flow in the network. Hence, both the definitions of the *multicommodity coding gain* g_ϕ and the *concurrent coding gain* g_λ are well-grounded. A value of g_ϕ or g_λ greater than 1 implies that the corresponding multicast network actually benefits from the deployment of network

coding techniques, while a value of g_ϕ or g_λ equal to 1 means that there is no incentive to implement any network coding process in the considered multicast network, with respect to this criterion. Finally, a value of g_ϕ or g_λ smaller than 1 means that network coding actually impedes the multicast network. The following pair of theorems basically states that this last situation can not occur.

Theorem 11. *Given a network $D = (V, A)$, set of commodities K with for each commodity k, source s^k, set of receivers R^k, and non-negative weight w^k, along with capacity c_a for each arc a, the weighted multicommodity coding gain g_ϕ is at least* 1.

The underlying idea of this theorem is again that routing using multicast only corresponds to the special case of using both multicast and network coding simultaneously in a network where no coding operation is performed but simple data duplication. Furthermore, the proof itself is very similar to the one of Theorem 3. The interested reader will find a formal proof of Theorem 11 in Appendix C.1.

Theorem 12. *Given a network $D = (V, A)$, set of commodities K with for each commodity k, source s^k, set of receivers R^k, and positive demand d^k, along with capacity c_a for each arc a, the concurrent coding gain g_λ is at least* 1.

The proof of Theorem 12 is very similar to the one of Theorem 11 and can also be found in Appendix C.1.

3.4.2 Features of the multicommodity coding gains

We shall now present various results regarding the multicommodity coding gains g_ϕ and g_λ. First notice that, in the single-commodity case, when the set K is a singleton, both g_ϕ and g_λ actually equal the coding gain g_φ.

3.4.2.1 Multiple-unicast

Consider the setting where, for each commodity k, there is only one single receiver r^k, $R^k = \{r^k\}$. As previously mentioned, in this particular case, a Steiner tree spanning both the source s^k and the receiver r^k is just a path between s^k and r^k. A multicommodity

Steiner flow then becomes a classical multicommodity flow. Furthermore a multicommodity coded flow also becomes a simple multicommodity flow since there is only one receiver per commodity. Those remarks immediately imply the two following lemmas:

Lemma 4. *Given a network $D = (V, A)$, set of commodities K with for each commodity k, source s^k, receiver r^k distinct from s^k, and non-negative weight w^k, along with capacity c_a for each arc a, it is possible to achieve the maximum weighted throughput among all commodities by performing forwarding operations only. Thus $g_\phi = 1$ in this setting.*

Lemma 5. *Given a network $D = (V, A)$, set of commodities K with for each commodity k, source s^k, receiver r^k distinct from s^k, and positive demand d^k, along with capacity c_a for each arc a, it is possible to achieve the maximum concurrent throughput between all commodities by performing forwarding operations only. Thus $g_\lambda = 1$ in this setting.*

Beware that, in order to get those two theorems, we still assume that inter-commodity coding is *not* allowed. Our approach hence differs from the multiple-unicast framework as it is traditionally defined in the network coding literature [61].

3.4.2.2 Coding gain and multicommodity coding gains

The following pair of theorems provides a connection between the single-commodity coding gain and its multicommodity counterparts.

Theorem 13. *Given a network $D = (V, A)$, set of commodities K with for each commodity k, source s^k, set of receivers R^k, and non-negative weight w^k, along with capacity c_a for each arc a, the multicommodity coding gain g_ϕ is upper-bounded by the quantity $|K| \max_{k \in K} g^k_\varphi$ where for each commodity k, g^k_φ is the coding gain associated to the instance with k as the only commodity in the network.*

Proof. For each commodity k, we denote by φ^k_S and φ^k_C respectively the value of a maximum Steiner flow, and the one of a maximum coded flow, when the whole network can be used to convey data from source s^k to set of receivers R^k. Hence, for each commodity k, $g^k_\varphi = \frac{\varphi^k_C}{\varphi^k_S}$. We first show that:

$$\frac{1}{|K|} \sum_{k \in K} w^k \varphi^k_S \leq \phi_S \tag{3.117}$$

For each commodity k, let x^k be a maximum Steiner flow in the single-commodity setting, so that $\varphi_S^k = \sum_{t \in \mathcal{T}^k} x_t^k$. Define multicommodity Steiner flow x by setting, for each commodity k and each Steiner tree $t \in \mathcal{T}^k$, $x_t = x_t^k$. For any arc a and any commodity k, the feasibility of Steiner flow x^k implies:

$$\sum_{t \in \mathcal{T}_a^k} x_t^k \leq c_a \tag{3.118}$$

so that for any arc a:

$$\sum_{k \in K} \sum_{t \in \mathcal{T}_a^k} x_t \leq |K| c_a \tag{3.119}$$

Furthermore, the weighted throughput of multicommodity Steiner flow x is:

$$\sum_{k \in K} w^k \sum_{t \in \mathcal{T}^k} x_t = \sum_{k \in K} w^k \sum_{t \in \mathcal{T}^k} x_t^k \tag{3.120}$$

$$= \sum_{k \in K} w^k \varphi_S^k \tag{3.121}$$

by optimality of Steiner flow x^k for each commodity k. Hence, scaling multicommodity Steiner flow x by a factor $\frac{1}{|K|}$ gives a feasible solution of value $\frac{1}{|K|} \sum_{k \in K} w^k \varphi_S^k$ to the maximum weighted multicommodity Steiner flow problem. This directly implies the desired inequality. Now, observe that $\phi_C \leq \sum_{k \in K} w^k \varphi_C^k$ since an optimal multicommodity coded flow induces a coded flow satisfying all capacity requirements for each commodity k. Combining this last inequality with Equation (3.117) yields:

$$g_\phi \leq \frac{|K| \sum_{k \in K} w^k \varphi_C^k}{\sum_{k \in K} w^k \varphi_S^k} \tag{3.122}$$

which can be rewritten as:

$$g_\phi \leq \frac{|K| \sum_{k \in K} w^k g_\varphi^k \varphi_S^k}{\sum_{k \in K} w^k \varphi_S^k} \tag{3.123}$$

thanks to the definition of g_φ^k for each commodity k. Bounding each g_φ^k by $\max_{k \in K} g_\varphi^k$ in the right-hand side of this last inequality provides the expected result. □

Theorem 14. *Given a network $D = (V, A)$, set of commodities K with for each commodity k, source s^k, set of receivers R^k, and positive demand d^k, along with capacity c_a for each arc a, the concurrent coding gain g_λ is upper-bounded by the quantity $|K| \max_{k \in K} g_\varphi^k$ where for each commodity k, g_φ^k is the coding gain associated to the instance with k as the only commodity in the network.*

Proof. Again, this proof is very similar to the one of Theorem 13. For each commodity k, we still denote by φ_S^k and φ_C^k respectively the value of a maximum Steiner flow, and the one of a maximum coded flow, when the whole network can be used to convey data from source s^k to set of receivers R^k. Hence, for each commodity k, $g_\varphi^k = \frac{\varphi_C^k}{\varphi_S^k}$. We first show that:

$$\frac{1}{|K|} \min_{k \in K} \frac{\varphi_S^k}{d^k} \leq \lambda_S \tag{3.124}$$

For each commodity k, let again x^k be a maximum Steiner flow in the single-commodity setting, so that $\varphi_S^k = \sum_{t \in \mathcal{T}^k} x_t^k$. Define multicommodity Steiner flow x as previously, by setting for each commodity k and each Steiner tree $t \in \mathcal{T}^k$, $x_t = x_t^k$, so that for any arc a, Equation (3.119) still holds. The minimum fraction of each demand which can be satisfied by multicommodity Steiner flow x is:

$$\min_{k \in K} \frac{\sum_{t \in \mathcal{T}^k} x_t}{d^k} = \min_{k \in K} \frac{\varphi_S^k}{d^k} \tag{3.125}$$

by optimality of Steiner flow x^k for each commodity k. Again, scaling multicommodity Steiner flow x by a factor $\frac{1}{|K|}$ yields a feasible solution to the maximum concurrent Steiner flow problem. This directly implies the desired inequality. We shall now justify that the following inequality holds:

$$\lambda_C \leq \min_{k \in K} \frac{\varphi_C^k}{d^k} \tag{3.126}$$

Observe that an optimal solution x to the maximum concurrent coded flow problem induces, for each commodity k, a coded flow satisfying all capacity requirements while ensuring:

$$\min_{r \in R^k} \sum_{p \in \mathcal{P}^{(k,r)}} x_p \leq \varphi_C^k \tag{3.127}$$

for any commodity k. (The multicommodity coded flow x induces a feasible coded flow with respect to each commodity.) This in turn implies:

$$\min_{k \in K} \frac{1}{d^k} \min_{r \in R^k} \sum_{p \in \mathcal{P}^{(k,r)}} x_p \leq \min_{k \in K} \frac{\varphi_C^k}{d^k} \tag{3.128}$$

which is the expected inequality by optimality of x. Combining this last result with Equation (3.124) leads to:

$$g_\lambda \leq \frac{|K| \min_{k \in K} \left(\varphi_C^k / d^k\right)}{\min_{k \in K} \left(\varphi_S^k / d^k\right)} \tag{3.129}$$

which can be rewritten as:

$$g_\lambda \leq \frac{|K| \min_{k \in K} \left(g_\varphi^k \varphi_S^k / d^k \right)}{\min_{k \in K} \left(\varphi_S^k / d^k \right)} \tag{3.130}$$

thanks to the definition of g_φ^k for each commodity k. Bounding g_φ^k by $\max_{k \in K} g_\varphi^k$ for each commodity k in the right-hand side of this last inequality concludes the proof. □

We shall now combine those two theorems with the results presented in the first chapter, so as to characterize the domain of each multicommodity coding gain for some networks.

3.4.2.3 Bidirected networks

Recall from Theorem 6 in the previous chapter that the coding gain g_φ of a bidirected network always equals 1. Combining this result with Theorem 13 and Theorem 14 respectively provides an upper-bound on the value of the multicommodity coding gain g_ϕ and the one of the concurrent coding gain g_λ of a bidirected network:

Corollary 4. *Given a bidirected network $B = (V, L)$, set of commodities K with for each commodity k, source s^k, set of receivers R^k, and non-negative weight w^k, along with capacity c_ℓ for each link ℓ, the multicommodity coding gain g_ϕ is upper-bounded by the number of commodities, $g_\phi \leq |K|$.*

Corollary 5. *Given a bidirected network $B = (V, L)$, set of commodities K with for each commodity k, source s^k, set of receivers R^k not containing s^k, and positive demand d^k, along with capacity c_ℓ for each link ℓ, the concurrent coding gain g_λ is upper-bounded by the number of commodities, $g_\lambda \leq |K|$.*

3.4.2.4 Undirected networks

Similarly, recall from Theorem 7 (in the previous chapter) that the coding gain g_φ of an undirected network lies in the interval $[1, 2]$. Combining this result with Theorem 13 and Theorem 14 respectively provides an upper-bound on the value of the multicommodity coding gain g_ϕ and the one of the concurrent coding gain g_λ of an undirected network:

Corollary 6. *Given an undirected network $G = (V, E)$, set of commodities K with for each commodity k, source s^k, set of receivers R^k not containing s^k, and non-negative weight w^k, along with capacity c_e for each edge e, the multicommodity coding gain g_ϕ is upper-bounded by twice the number of commodities, $g_\phi \leq 2|K|$.*

Corollary 7. *Given an undirected network $G = (V, E)$, set of commodities K with for each commodity k, source s^k, set of receivers R^k not containing s^k, and positive demand d^k, along with capacity c_e for each edge e, the concurrent coding gain g_λ is upper-bounded by twice the number of commodities, $g_\lambda \leq 2|K|$.*

3.4.2.5 Directed networks

From Theorem 8, the coding gain g_φ of a directed network can be arbitrarily large. As previously mentioned, in the special case of a single-commodity, the multicommodity coding gain g_ϕ and the concurrent coding gain g_λ are both equal to g_φ. Therefore, those two quantities can be arbitrarily large in a directed network, as emphasized by the two theorems below:

Theorem 15. *For any real number $\eta \geq 1$, there exists an infinite family of directed networks with unit capacity and a single-commodity such that the ratio between the value of a maximum weighted multicommodity coded flow and the one of a maximum weighted multicommodity Steiner flow satisfies $g_\phi \geq \eta$.*

Theorem 16. *For any real number $\eta \geq 1$, there exists an infinite family of directed networks with unit capacity and a single-commodity such that the ratio between the value of a maximum concurrent coded flow and the one of a maximum concurrent Steiner flow satisfies $g_\lambda \geq \eta$.*

3.4.3 Experimental evaluation of the multicommodity coding gains

3.4.3.1 Setting

We use fifteen network topologies taken from the *SNDlib library* [97]. We then arbitrarily build a set K of commodities by picking for each commodity k a vertex s^k as source, and a subset of vertices R^k as receivers. The capacity of each channel is picked

in the set [10] (of all integers between 1 and 10) uniformly at random. When dealing with multicommodity weighted information flows, the weight of each commodity k is set to $w^k = \big(|R^k| / \sum_{k \in K} |R^k|\big)$. As far as concurrent information flows are concerned, the demand d^k of each commodity k is picked in the set $\{1, 2, 5\}$ uniformly at random.

We then compute each optimal Steiner flow by coupling a column generation algorithm with a subroutine generating minimum-cost Steiner trees by solving to optimality the mixed-integer linear program 2.34 presented in the previous chapter (see also [55]). Each optimal coded flow problem is computed by solving its associated arc formulation thanks to a linear programming solver. All algorithms are implemented in Julia 0.3.8 [98, 99]. We use the Julia package *JuMP* [100] to call the open-source (mixed-integer) linear programming solver *CLP/CBC* [101].

3.4.3.2 Undirected networks

Our test bench is made of fifteen networks whose features are summarized in Table 3.1. For each instance, we give the number of vertices $|V|$, the number of edges $|E|$, the number of commodity $|K|$, and the list of the numbers of receivers per commodity. We also provide the list of weights per commodity along with the list of demands per commodity.

Instance	$\|V\|$	$\|E\|$	$\|K\|$	$(\|R^k\|)_{k \in K}$	$(w^k)_{k \in K}$	$(d^k)_{k \in K}$
abilene	12	15	2	3 ; 3	1/2 ; 1/2	1 ; 1
atlanta	15	22	2	4 ; 5	4/9 ; 5/9	2 ; 1
france	25	45	2	4 ; 6	2/5 ; 3/5	2 ; 1
geant	22	36	3	3 ; 4 ; 10	3/17 ; 4/17 ; 10/17	5 ; 2 ; 2
germany50	50	88	3	4 ; 4 ; 4	1/3 ; 1/3 ; 1/3	1 ; 2 ; 2
giul39	39	86	4	3 ; 3 ; 3 ; 3	1/4 ; 1/4 ; 1/4 ; 1/4	1 ; 2 ; 1 ; 1
india35	35	80	3	4 ; 4 ; 4	1/3 ; 1/3 ; 1/3	1 ; 2 ; 2
newyork	16	49	2	4 ; 4	1/2 ; 1/2	1 ; 5
nobel-eu	28	41	3	3 ; 3 ; 4	3/10 ; 3/10 ; 2/5	1 ; 2 ; 2
norway	27	51	3	3 ; 4 ; 5	1/4 ; 1/3 ; 5/12	5 ; 2 ; 2
pioro40	40	89	2	3 ; 6	1/3 ; 2/3	1 ; 1
polska	12	18	2	3 ; 5	3/8 ; 5/8	1 ; 2
ta1	24	55	2	4 ; 6	2/5 ; 3/5	1 ; 2
ta2	65	108	3	3 ; 4 ; 5	1/4 ; 1/3 ; 5/12	1 ; 5 ; 1
zib54	54	81	2	4 ; 4	1/2 ; 1/2	1 ; 2

Table 3.1 – Some features of the fifteen instances.

Table 3.2 gives, for each undirected instance, information regarding the maximum weighted multicommodity Steiner flow returned by the column generation algorithm. More precisely, column ϕ_S indicates the value of the optimal Steiner flow. Column *Trees* gives the number of trees actually used by the optimal Steiner flow to convey some positive flow. Finally, column *Iterations* gives the number of iterations made by the column generation algorithm while column *Time* provides the associated running time expressed in seconds.

Instance	ϕ_S	Trees	Iterations	Time (s)
abilene	6.0	4	2	0.34
atlanta	6.07	9	21	1.54
france	5.13	6	15	1.34
geant	5.14	6	14	2.53
germany50	6.17	27	16	4.35
giul39	9.75	54	30	8.1
india35	7.33	22	17	2.49
newyork	15.5	15	16	1.14
nobel-eu	6.0	8	11	0.65
norway	9.06	30	26	6.17
pioro40	11.07	32	56	125.38
polska	5.33	11	12	0.68
ta1	12.88	22	28	2.67
ta2	9.11	27	18	4.24
zib54	6.25	10	15	1.88

Table 3.2 – Features of the maximum weighted multicommodity Steiner flow returned by the column generation algorithm for each undirected network.

Table 3.3 is very similar to Table 3.2 in that it provides, for each undirected instance, information regarding the coded flow returned by the linear programming solver. Column ϕ_C gives the value of the optimal coded flow while column *Time* provides the associated running time expressed in seconds.

Observe that, for each undirected instance, the multicommodity coding gain g_ϕ equals 1.

Table 3.4 gives, for each undirected instance, information regarding the maximum concurrent Steiner flow returned by the column generation algorithm. More precisely, column λ_S indicates the value of the optimal Steiner flow. Column *Trees* gives the number of trees actually used by the optimal Steiner flow to convey some positive flow. Finally, column

Instance	ϕ_C	Time (s)
abilene	6.0	0.01
atlanta	6.07	0.01
france	5.13	0.03
geant	5.14	0.04
germany50	6.17	0.4
giul39	9.75	0.27
india35	7.33	0.11
newyork	15.5	0.03
nobel-eu	6.0	0.04
norway	9.06	0.09
pioro40	11.07	0.08
polska	5.33	0.01
ta1	12.88	0.05
ta2	9.11	0.26
zib54	6.25	0.05

Table 3.3 – Features of the maximum weighted multicommodity coded flow returned by the linear programming solver for each undirected network.

Iterations gives the number of iterations made by the column generation algorithm while column *Time* provides the associated running time expressed in seconds.

Instance	λ_S	Trees	Iterations	Time (s)
abilene	6.0	4	4	0.43
atlanta	3.67	8	8	0.58
france	3.0	6	13	1.28
geant	0.89	7	9	2.17
germany50	3.5	24	22	5.45
giul39	7.0	38	29	4.39
india35	4.4	20	26	4.04
newyork	5.17	22	24	1.64
nobel-eu	3.4	7	10	0.66
norway	2.61	25	29	5.41
pioro40	10.29	35	44	37.86
polska	1.5	3	2	0.2
ta1	7.67	19	31	2.66
ta2	3.17	19	27	5.04
zib54	4.17	11	16	2.15

Table 3.4 – Features of the maximum concurrent Steiner flow returned by the column generation algorithm for each undirected network.

Table 3.5 presents, for each undirected instance, information regarding the coded flow

returned by the linear programming solver. Column λ_C gives the value of the optimal coded flow while column *Time* provides the associated running time expressed in seconds.

Instance	λ_C	Time (s)
abilene	6.0	0.02
atlanta	3.67	0.01
france	3.0	0.03
geant	0.89	0.04
germany50	3.5	0.18
giul39	7.0	0.15
india35	4.4	0.14
newyork	5.17	0.03
nobel-eu	3.4	0.03
norway	2.61	0.08
pioro40	10.29	0.09
polska	1.5	0.01
ta1	7.67	0.04
ta2	3.17	0.11
zib54	4.17	0.12

Table 3.5 – Features of the maximum concurrent coded flow returned by the linear programming solver for each undirected network.

Notice that, for each undirected instance, the concurrent coding gain g_λ is equal to 1.

3.4.3.3 Bidirected networks

The test bench for bidirected instances is obtained by considering the bidirected network associated to each undirected instance presented in Table 3.1. All other features of the undirected instances remain unchanged.

Table 3.6 provides, for each bidirected instance, information regarding the maximum weighted multicommodity Steiner flow returned by the column generation algorithm. More precisely, column ϕ_S indicates the value of the optimal Steiner flow. Column *Trees* gives the number of trees actually used by the optimal Steiner flow to convey some positive flow. Finally, column *Iterations* gives the number of iterations made by the column generation algorithm while column *Time* provides the associated running time expressed in seconds.

Table 3.7 gives, for each bidirected instance, information regarding the coded flow returned by the linear programming solver. Column ϕ_C gives the value of the optimal

Instance	ϕ_S	Trees	Iterations	Time (s)
abilene	7.5	4	4	0.35
atlanta	6.29	6	15	0.4
france	5.12	10	15	0.65
geant	5.14	5	10	0.68
germany50	9.0	22	24	2.49
giul39	11.5	46	29	2.86
india35	14.33	51	46	4.16
newyork	18.0	19	16	0.65
nobel-eu	9.14	16	10	0.42
norway	10.44	15	21	1.3
pioro40	14.71	29	34	2.92
polska	5.67	8	8	0.15
ta1	12.88	17	23	1.25
ta2	10.78	29	21	2.76
zib54	8.0	11	11	0.79

Table 3.6 – Features of the maximum weighted multicommodity Steiner flow returned by the column generation algorithm for each bidirected network.

coded flow while column *Time* provides the associated running time expressed in seconds.

Instance	ϕ_C	Time (s)
abilene	7.5	0.01
atlanta	6.29	0.01
france	5.13	0.03
geant	5.14	0.04
germany50	9.0	0.17
giul39	11.5	0.2
india35	14.33	0.16
newyork	18.0	0.03
nobel-eu	9.14	0.08
norway	10.44	0.06
pioro40	14.71	0.06
polska	5.67	0.01
ta1	12.88	0.04
ta2	10.78	0.09
zib54	8.0	0.04

Table 3.7 – Features of the maximum weighted multicommodity coded flow returned by the linear programming solver for each bidirected network.

Observe that, for each bidirected instance, the multicommodity coding gain g_ϕ always equals 1.

Table 3.8 gives, for each bidirected instance, information regarding the maximum concurrent Steiner flow returned by the column generation algorithm. More precisely, column λ_S indicates the value of the optimal Steiner flow. Column *Trees* gives the number of trees actually used by the optimal Steiner flow to convey some positive flow. Finally, column *Iterations* gives the number of iterations made by the column generation algorithm while column *Time* provides the associated running time expressed in seconds.

Instance	λ_S	Trees	Iterations	Time (s)
abilene	6.0	2	3	0.4
atlanta	3.67	4	5	0.15
france	3.0	7	7	0.38
geant	0.89	7	7	0.62
germany50	3.5	21	23	2.58
giul39	7.0	33	32	3.3
india35	7.33	38	41	4.03
newyork	6.0	17	28	1.14
nobel-eu	4.0	6	8	0.38
norway	3.0	19	23	1.54
pioro40	14.0	33	48	4.41
polska	1.5	2	2	0.05
ta1	7.67	17	25	1.36
ta2	3.17	16	29	4.15
zib54	5.33	13	15	1.09

Table 3.8 – Features of the maximum concurrent Steiner flow returned by the column generation algorithm for each bidirected network.

Table 3.9 presents, for each bidirected instance, information regarding the coded flow returned by the linear programming solver. Column λ_C gives the value of the optimal coded flow while column *Time* provides the associated running time expressed in seconds.

Observe that, for each bidirected instance, the concurrent coding gain g_λ equals 1.

3.4.3.4 Directed networks

Each directed instance below is actually a bidirected network where, for each link (pair of reverse arcs), the capacity of one of the two arcs of the link is multiplied by a coefficient picked in the set $\{1, 2, 3, 5, 10\}$ uniformly at random. This choice preserve the original network connectivity while removing some restrictions found in the previous bidirected

Instance	λ_C	Time (s)
abilene	6.0	0.01
atlanta	3.67	0.01
france	3.0	0.03
geant	0.89	0.04
germany50	3.5	0.09
giul39	7.0	0.15
india35	7.33	0.12
newyork	6.0	0.02
nobel-eu	4.0	0.03
norway	3.0	0.06
pioro40	14.0	0.07
polska	1.5	0.01
ta1	7.67	0.04
ta2	3.17	0.1
zib54	5.33	0.04

Table 3.9 – Features of the maximum concurrent coded flow returned by the linear programming solver for each bidirected network.

setting. All other features of the bidirected instances remain unchanged.

Table 3.10 provides, for each directed instance, information regarding the maximum weighted multicommodity Steiner flow returned by the column generation algorithm. More precisely, column ϕ_S indicates the value of the optimal Steiner flow. Column *Trees* gives the number of trees actually used by the optimal Steiner flow to convey some positive flow. Finally, column *Iterations* gives the number of iterations made by the column generation algorithm while column *Time* provides the associated running time expressed in seconds.

Table 3.11 gives, for each directed instance, information regarding the coded flow returned by the linear programming solver. Column ϕ_C gives the value of the optimal coded flow while column *Time* provides the associated running time expressed in seconds.

Notice that, for each directed instance, the multicommodity coding gain g_ϕ always stays equal to 1.

Table 3.12 gives, for each directed instance, information regarding the maximum concurrent Steiner flow returned by the column generation algorithm. More precisely, column λ_S indicates the value of the optimal Steiner flow. Column *Trees* gives the number of trees actually used by the optimal Steiner flow to convey some positive flow. Finally, column

Instance	ϕ_S	Trees	Iterations	Time (s)
abilene	7.5	4	3	0.34
atlanta	6.29	6	11	0.31
france	5.63	5	10	0.45
geant	9.64	11	14	0.93
germany50	11.67	33	26	2.78
giul39	11.5	32	24	2.36
india35	23.67	48	29	2.58
newyork	27.5	20	18	0.73
nobel-eu	9.57	11	7	0.3
norway	15.89	19	27	1.71
pioro40	16.14	29	25	2.11
polska	5.67	6	6	0.12
ta1	14.38	13	25	1.34
ta2	10.89	22	18	2.38
zib54	15.0	16	13	0.94

Table 3.10 – Features of the maximum weighted multicommodity Steiner flow returned by the column generation algorithm for each directed network.

Instance	ϕ_C	Time (s)
abilene	7.5	0.03
atlanta	6.29	0.01
france	5.63	0.03
geant	9.64	0.04
germany50	11.67	0.14
giul39	11.5	0.17
india35	23.67	0.08
newyork	27.5	0.09
nobel-eu	9.57	0.03
norway	15.89	0.06
pioro40	16.14	0.06
polska	5.67	0.01
ta1	14.38	0.04
ta2	10.89	0.08
zib54	15.0	0.04

Table 3.11 – Features of the maximum weighted multicommodity coded flow returned by the linear programming solver for each bidirected network.

Iterations gives the number of iterations made by the column generation algorithm while column *Time* provides the associated running time expressed in seconds.

Table 3.13 presents, for each directed instance, information regarding the coded flow

Instance	λ_S	Trees	Iterations	Time (s)
abilene	6.0	2	3	0.37
atlanta	3.67	4	4	0.12
france	3.0	4	6	0.31
geant	1.86	13	10	0.75
germany50	6.5	30	25	2.81
giul39	9.2	36	34	3.57
india35	11.0	36	46	4.48
newyork	9.17	21	33	1.35
nobel-eu	4.5	6	10	0.45
norway	4.2	15	17	1.19
pioro40	15.5	26	30	2.84
polska	1.5	2	2	0.06
ta1	7.67	14	17	0.99
ta2	3.17	12	19	2.7
zib54	10.0	19	25	1.82

Table 3.12 – Features of the maximum concurrent Steiner flow returned by the column generation algorithm for each directed network.

returned by the linear programming solver. Column λ_C gives the value of the optimal coded flow while column *Time* provides the associated running time expressed in seconds.

Instance	λ_C	Time (s)
abilene	6.0	0.01
atlanta	3.67	0.01
france	3.0	0.03
geant	1.86	0.05
germany50	6.5	0.18
giul39	9.2	0.24
india35	11.0	0.07
newyork	9.17	0.02
nobel-eu	4.5	0.02
norway	4.2	0.05
pioro40	15.5	0.06
polska	1.5	0.01
ta1	7.67	0.04
ta2	3.17	0.15
zib54	10.0	0.04

Table 3.13 – Features of the maximum concurrent coded flow returned by the linear programming solver for each directed network.

For each directed instance, the concurrent coding gain g_λ stays equal to 1.

3.4.3.5 Analysis

For each type of information flow, and for each instance, the proposed algorithm finds an optimal solution in a matter of seconds. Regardless of the network structure, it seems there is few incentives to use coding techniques, as far as the multicommodity coding gains are concerned. However, in a multicommodity setting, the decentralized approach allowed by the use of coding techniques may favor network coding over multicast alone. Finally, we observe that, the value of a maximum concurrent information flow is lower than the one of a maximum weighted multicommodity information flow (beware that the two settings slightly differs). This can be interpreted as the price of fairness.

3.5 Conclusion

In this chapter, we explained how to extend the framework of information flows to the case where multiple sessions or commodities are simultaneously accessing the network. We showed that given the proper definition of a *multicommodity Steiner flow* along with the one of a *multicommodity coded flow*, it is possible to define, model, and compute an information flow variant of some classical multicommodity flow problems as they appear in network optimization. We proposed to extend the notion of coding gain to the multicommodity setting, so as to define an indicator, called the *multicommodity coding gain*, allowing comparisons between multicast alone and multicast combined with coding techniques. By using well-known features of the coding gain in a single commodity setting, we analysed the *domain* of values one can expect the multicommodity coding gain to take. Those results are summarized in Table 3.14 below. Recall that the greater the value actually taken by the coding gain in a given multicast network, the more beneficial network coding techniques would be if implemented in this particular network.

Structure	Coding gain	Multicommodity coding gain	Concurrent coding gain
Bidirected	$\{1\}$	$[1, \|K\|]$	$[1, \|K\|]$
Undirected	$[1, 2]$	$[1, 2\|K\|]$	$[1, 2\|K\|]$
Directed	$[1, +\infty[$	$[1, +\infty[$	$[1, +\infty[$

Table 3.14 – Coding gain domain for each channel model. $|K|$ is the number of commodities.

Table 3.14 also highlights how impactful the channel model actually is on both multicommodity coding gains. Observe that the multicommodity setting seems to dilate the *potential* of network coding with respect to the single commodity setting. However, we would like to stress out that we ignore whether those bounds are actually tight.

Similarly to what we did in this chapter, it is possible to use the present framework of multicommodity information flows to extend the classical minimum-cost multicommodity flow problem. We refer the interested reader to the paper by Raayatpanah et al. [102] for an in-depth presentation of the network coding variant of this problem.

Notice that, in the whole chapter, we always assumed that a single operator would take all decisions regarding the network use. It would be interesting to consider instead the game-theoretic variant of each problem, where each commodity is selfishly trying to satisfy its own needs regardless of those of the others.

Finally, as already explained, we focus on a setting where inter-commodity coding is explicitly prohibited. We would like to mention that handling inter-commodity coding remains an open and difficult research topic, even when restricted to the special case where each commodity involves only one single receiver, known as the *multiple unicast* setting in the network coding literature.

Chapter 4

Convex cost information flows

4.1 Introduction

4.1.1 Motivation

The multicast framework was primarily developed to cope with the increasing demand for live services, like video-streaming, teleconferencing, or online gaming. It is then natural to study the problem of designing a multicast routing scheme at low cost. Furthermore, one may ask whether network coding might help to reduce the congestion in a multicast network.

Previous works on this topic mainly focused on the so-called multicast congestion problem [103, 104, 96] where one wishes to minimize the maximum over the channels of the ratios "flow over available bandwidth". In this chapter, we study a different problem where each channel of the network has its own convex cost function and we want to minimize the sum over the channels of the cost contribution of each channel.

4.1.2 Content

We first present the so-called *minimum convex cost flow problem* which will be the foundation for all subsequent generalizations. The second problem we shall present, referred to as the *minimum convex cost Steiner flow problem*, is one of those possible generalizations of the previous problem, with Steiner trees replacing paths as the basic component of the flow. We provide a complete study of the minimum convex cost Steiner flow problem including an algorithmic perspective. The third problem we consider, namely the *minimum*

convex cost coded flow problem, naturally appears when one is trying to reduce the cost of conveying data in a multicast network by using coding techniques. We shall then introduce the notion of *cost coding gain* as an indicator of the benefit one can expect from using those coding techniques in a multicast network to tackle cost issues.

4.1.3 Minimum convex cost flow

4.1.3.1 Problem statement

In the following, we deal with convex sets and convex functions. Recall that a set X of a real vector space is *convex* if, for any pair of points x_1 and x_2 in X, and any γ in the interval $[0, 1]$, the point $(1-\gamma)x_1+\gamma x_2$ lies in X. Furthermore, a function ψ from a convex set X to $\mathbb{R}$ is *convex* if, for any pair of points x_1 and x_2 in X, and any γ in the interval $[0, 1]$, the inequality $\psi((1-\gamma)x_1 + \gamma x_2)) \leq (1-\gamma)\psi(x_1) + \gamma\psi(x_2)$ holds.

For the sake of simplicity, we focus again on the case of directed networks, without loss of generality since it encompasses the case of undirected ones. The instance is made of a network $D = (V, A)$ with a special node s called the *source*, another peculiar node r referred to as the *receiver*, and a positive *demand* d modeling an amount of flow to be conveyed from the source to the receiver. We are also given, for each arc a, a convex, continuously differentiable, non-decreasing, and univariate *cost function* ψ_a from $\mathbb{R}_+$ to $\mathbb{R}_+$. For each arc a, the cost function ψ_a may be parameterized by a capacity c_a which is a given feature of the arc. For each arc a, we denote by ψ'_a the first derivative of ψ_a. We shall further assume that given any non-negative real number z, we can compute (via an oracle) the values $\psi_a(z)$ and $\psi'_a(z)$ in polynomial time for any prescribed accuracy. Recall that a *flow* x is an assignment of a non-negative real value x_p to each simple path p from the source to the receiver. We denote by $\mathcal{P}$ the set of such paths, and by $\mathcal{P}_a$ the subset of paths in $\mathcal{P}$ using arc a. Given a flow x, we call x_p the *flow* on path p, and we define f_a as the induced *flow* on arc a, $f_a = \sum_{p\in\mathcal{P}_a} x_p$. The associated vector f is called the *arc projection* of flow x. Given a capacity vector c on the arcs, a flow x satisfies the capacity requirements if, for each arc a, the amount of flow on a is smaller than, or equal to, the arc capacity, namely $f_a \leq c_a$. Moreover, flow x enforces the demand if the receiver actually experiences a throughput of value d, $\sum_{p\in\mathcal{P}} x_p = d$. The problem we consider here is to

find a flow x, enforcing the demand, and minimizing the *overall cost function* ψ defined as the sum of the cost functions $(\psi_a)_{a \in A}$ over all arcs:

$$\psi(x) = \sum_{a \in A} \psi_a(f_a) \tag{4.1}$$

where vector f is the arc projection of flow x. Notice that ψ is also convex, non-decreasing, and defined from $\mathbb{R}_+^{\mathcal{P}}$ to $\mathbb{R}_+$, thanks to the previous assumptions on the family of cost functions $(\psi_a)_{a \in A}$. Given a flow x, we shall refer to the quantity $\psi(x)$ as the *overall cost* of x, which, from our previous assumptions, can be computed in polynomial time. Also notice that this overall cost is separable with respect to the arcs but not with respect to the paths. Further observe that, in the design of a flow, there is no incentive to route more than d units of flow on any arc since the total demand is d and ψ_a is non-decreasing for each arc a. We further assume that $\psi_a(0) = 0$ for each arc a. This can be done without loss of generality, since any fixed cost incurred by the use of the network can be kept aside from the objective function. Finally, it is possible to make the capacity requirements implicit by assuming that for each arc a such that $c_a \leq d$, the penalty $\psi_a(f_a)$ incurred by routing a flow of value $f_a \in [c_a, d]$ is high enough so as to prevent any capacity violation. This problem transformation can be achieved by taking for each arc a a suitable function ψ_a, which increases quickly at the neighborhood of c_a. We provide an example of such a family of functions later in this chapter. Although this last assumption is not mandatory for the problem at hand, it will be useful later on. In the following, we denote by $\mathcal{X}_{\mathcal{P}}$ the set of all flows enforcing the demand constraint:

$$\mathcal{X}_{\mathcal{P}} = \left\{ x \in \mathbb{R}_+^{\mathcal{P}} : \sum_{p \in \mathcal{P}} x_p = d \right\} \tag{4.2}$$

A flow x is *feasible* if $x \in \mathcal{X}_{\mathcal{P}}$. We are now ready to formally express our problem:

Problem	*Minimum convex cost (uncapacitated) flow*
Instance	Network $D = (V, A)$, source s, receiver r distinct from s, convex cost function ψ_a for each arc a, positive demand d
Solution	A feasible flow $x \in \mathcal{X}_{\mathcal{P}}$ in D
Objective	Minimize the overall cost $\psi(x)$ of flow x as defined in Equation (4.1)

We denote by Ψ_F the optimal value of this problem for the instance of interest:

$$\Psi_F = \min_{x \in \mathcal{X}_\mathcal{P}} \psi(x) \tag{4.3}$$

4.1.3.2 Mathematical programming formulation

The minimum convex cost flow problem can be formalized using mathematical programming:

$$\begin{cases} \Psi_F = \min \sum_{a \in A} \psi_a(f_a) & (4.4) \\ \text{s.t.} \quad \sum_{p \in \mathcal{P}} x_p = d & (4.5) \\ \qquad f_a \geq \sum_{p \in \mathcal{P}_a} x_p \quad \forall a \in A & (4.6) \\ \qquad f_a, x_p \geq 0 \quad \forall a \in A, p \in \mathcal{P} & (4.7) \end{cases}$$

For each path p, variable x_p stands for the amount of flow routed on p, while, for each arc a, variable f_a models the amount of flow going through a. Observe that, since we may have an exponential number of paths in $\mathcal{P}$, the previous model may also have an exponential number of variables. Constraint (4.5) ensures that flow x satisfies the demand d. Combined with the assumption that each function ψ_a is non-decreasing and the fact that the objective function is to be minimized, Constraints (4.6) enforce variable f_a to take the value $\sum_{p \in \mathcal{P}_a} x_p$, for any arc a, in an optimal solution.

This path formulation may involve a huge number of variables. Fortunately, the following polynomial size formulation of the minimum convex cost flow problem can be used instead:

$$\begin{cases} \Psi_F = \min \sum_{a \in A} \psi_a(f_a) & (4.8) \\ \text{s.t.} \sum_{a \in \delta^+(v)} f_a - \sum_{a \in \delta^-(v)} f_a = b_v(d) \quad \forall v \in V & (4.9) \\ \qquad f_a \geq 0 \quad \forall a \in A & (4.10) \end{cases}$$

with

$$b_v(d) = \begin{cases} d & \text{if } v = s \quad (4.11) \\ -d & \text{if } v = r \quad (4.12) \\ 0 & \text{otherwise} \quad (4.13) \end{cases}$$

where for each arc a, variable f_a stands for the amount of flow going through a.

4.1.3.3 Complexity and algorithms

Thanks to this arc formulation, the minimum convex cost flow problem can be viewed as a problem of minimizing a convex function over a convex set induced by a *polynomial number* of linear constraints. Therefore, all standard techniques of convex optimization are available, see the book by Boyd and Vandenberghe [105] for extensive results regarding this field. Notice that, in the arc formulation of the minimum convex cost flow problem, we could directly incorporate the capacity requirements to the set of constraints, instead of assuming them implicitly through the cost functions, while keeping a tractable problem. This harder variant of the problem can be efficiently solved by various algorithms, see the survey by Ouorou et al. [106]. Also have a look at the chapter dealing with this topic in the classical book by Ahuja et al. [7]. The interested reader may also review the book by Bertsekas et al [107].

4.2 Minimum convex cost Steiner flow

4.2.1 Problem statement

The setting of this problem is similar to the one of the minimum convex cost flow problem, except that the single receiver is replaced by a set of receiver vertices. For brevity, we still focus the study on the case of directed networks (without loss of generality since it still encompasses the case of undirected ones). Formally, the instance is made of a *network* $D = (V, A)$ with a *source* s, a set R of *receivers* distinct from s, and a positive *demand* d. Recall that a *Steiner flow* x is an assignment of a non-negative real value x_t to each Steiner tree t spanning both s and R. We denote by $\mathcal{T}$ the set of such Steiner trees, and by $\mathcal{T}_a$ the subset of trees in $\mathcal{T}$ using arc a. Given a Steiner flow x, we call x_t the *flow* on tree t and we define f_a as the induced *flow* on arc a, $f_a = \sum_{t \in \mathcal{T}_a} x_t$. The associated vector f will be referred to as the *arc projection* of Steiner flow x. Given a capacity vector c on the arcs, a Steiner flow x satisfies the capacity requirements if, for each arc a, the amount of flow on a is smaller than, or equal to, the arc capacity, namely $f_a \leq c_a$. Furthermore, we say that x meets the demand if each receiver experiences a throughput of value d, namely $\sum_{t \in \mathcal{T}} x_t = d$. When talking about minimum convex cost Steiner flow, a natural

setting would be as follows: given a convex *cost function* ψ from $\mathbb{R}_+^{\mathcal{T}}$ to $\mathbb{R}_+$, and a capacity vector c on the arcs, find a Steiner flow x, satisfying both the capacity requirements and the demand constraint, whose cost with respect to ψ is minimum. Unfortunately, it is NP-hard to decide whether there exists a *feasible* solution to this problem, since finding a point in the polytope of constraints amounts to solving the decision version of the problem of fractionally packing Steiner trees, which is NP-hard [40]. Hence, in this chapter, we focus on a slightly different problem where we relax the capacity requirements. Like in the previous study, we shall assume that the overall cost function ψ is of the form provided in Equation (4.1), namely $\psi(x) = \sum_{a \in A} \psi_a(f_a)$, where vector f is the arc projection of Steiner flow x, and, for each arc a, ψ_a is again a convex, continuously differentiable, non-decreasing, and univariate *cost function* from $\mathbb{R}_+$ to $\mathbb{R}_+$ with $\psi_a(0) = 0$. For each arc a, we still denote by ψ'_a the first derivative of ψ_a. We shall again assume that given any non-negative real number z, the values $\psi_a(z)$ and $\psi'_a(z)$ can be computed in polynomial time for any prescribed accuracy. In order to keep the problem tractable, while taking into accounts the existence of a capacity c_a on each arc a of the network, it is desirable to pick, for each arc a such that $c_a \leq d$, a cost function ψ_a which increases quickly at the neighborhood of c_a, so as to prevent any practical capacity violation. Given a Steiner flow x, its arc projection f, and an arc a, we call $\psi_a(f_a)$ the *cost induced by x on arc a*, while the *overall cost of x* is the quantity $\psi(x)$. Notice that this overall cost is separable with respect to the arcs but not with respect to the trees. In the following, we denote by $\mathcal{X}_{\mathcal{T}}$ the set of all Steiner flows meeting the demand constraint:

$$\mathcal{X}_{\mathcal{T}} = \left\{ x \in \mathbb{R}_+^{\mathcal{T}} : \sum_{t \in \mathcal{T}} x_t = d \right\} \tag{4.14}$$

A Steiner flow x is *feasible* if $x \in \mathcal{X}_{\mathcal{T}}$. We are now ready to formally express our problem:

Problem	*Minimum convex cost (uncapacitated) Steiner flow*
Instance	Network $D = (V, A)$, source s, set of receivers R, convex cost function ψ_a for each arc a, positive demand d
Solution	A feasible Steiner flow $x \in \mathcal{X}_{\mathcal{T}}$ in D
Objective	Minimize the overall cost $\psi(x)$ of Steiner flow x as defined in Equation (4.1)

We denote by ψ_S the value of a minimum-cost Steiner flow for the considered instance:

$$\psi_S = \min_{x \in \mathcal{X}_\mathcal{T}} \psi(x) \tag{4.15}$$

When the set R is a singleton, $R = \{r\}$, a Steiner tree spanning s and R is a simple path between the source and the unique receiver r. Hence, the minimum convex cost Steiner flow problem can indeed be regarded as a generalization of the minimum convex cost flow problem to more than one receivers.

Consider the special case where all vertices except the source are receivers, $R = V \setminus \{s\}$. Observe that, in this setting, a Steiner tree spanning s and R is actually a spanning tree and we can define a *spanning flow* accordingly. We naturally refer to the associated problem as the *minimum convex cost spanning flow problem*:

Problem	*Minimum convex cost (uncapacitated) spanning flow*
Instance	Network $D = (V, A)$, source s, convex cost function ψ_a for each arc a positive demand d
Solution	A feasible spanning flow $x \in \mathcal{X}_\mathcal{T}$ in D
Objective	Minimize the overall cost $\psi(x)$ of spanning flow x as defined in Equation (4.1)

We still denote by ψ_S the value of a minimum-cost spanning flow for the considered instance, since the meaning should be clear from the context.

4.2.2 A detailed example

We consider again the directed butterfly network which is depicted on Figure 4.1 a). We are interested in solving the minimum cost Steiner flow problem under unit demand, namely $d = 1$, while the cost function ψ_a on each arc a is given by the expression $\psi_a(z) = \frac{z}{c_a - z}$ for any $z \in [0, d]$, where the capacity c_a of each arc a is set to 1. A minimum-cost Steiner flow $\bowtie$ is depicted on Figure 4.1 b). Notice that it uses five Steiner trees to satisfy the whole demand. The overall cost induced by Steiner flow $\bowtie$ is 10.97.

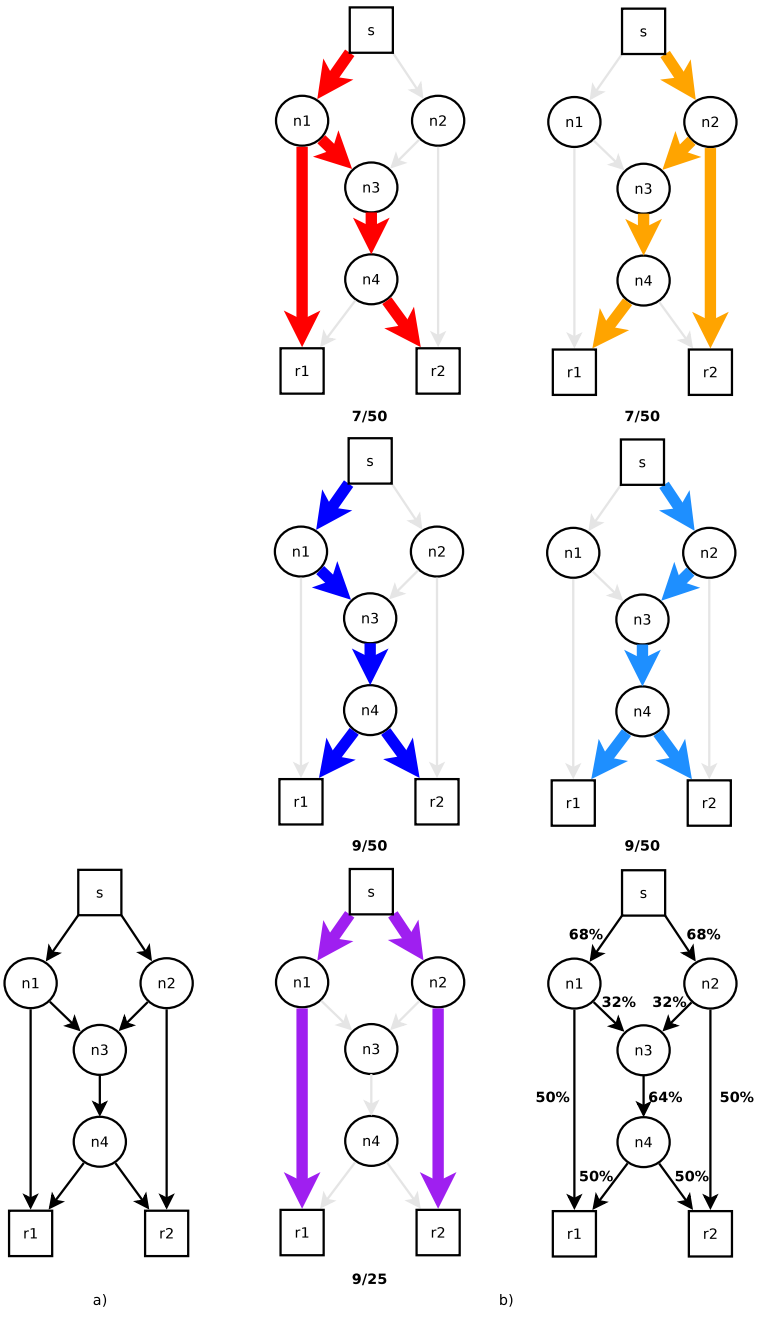


Figure 4.1 – a) The directed butterfly network. b) A minimum-cost Steiner flow $\not{x}$ using five trees in the directed butterfly network. Each tree on the first row routes 7/50 units of flow, while each tree on the second row conveys 9/50 units of flow. The last tree on the third row brings 9/25 units of flow to each receiver. The network at the bottom-right provides the percentage of the load $\frac{f_a}{c_a}$ for each arc a. The cost induced by Steiner flow $\not{x}$ is 10.97.

4.2.3 Mathematical programming formulation

The minimum convex cost Steiner flow problem can be formalized using mathematical programming:

$$\begin{cases} \psi_S = \min \sum_{a \in A} \psi_a \left(\sum_{t \in \mathcal{T}_a} x_t \right) & (4.16) \\ \text{s.t.} \quad \sum_{t \in \mathcal{T}} x_t = d & (4.17) \\ \qquad x_t \geq 0 \quad \forall t \in \mathcal{T} & (4.18) \end{cases}$$

For each Steiner tree t, the variable x_t stands for the amount of flow routed on t. Observe that, since we may have an exponential number of trees in $\mathcal{T}$, this model may also have an exponential number of variables. Constraint (4.17) ensures that the Steiner flow x meets the demand d.

4.2.4 Complexity and algorithms

Although we have relaxed the capacity requirements, the resulting problem is still intractable as emphasized by the next theorem:

Theorem 17. *The minimum convex cost Steiner flow problem remains NP-hard even if the cost function of each arc is a linear mapping.*

Proof. Assume that, for each arc a, the cost function ψ_a is of the form $\psi_a(z) = w_a z$ for any non-negative real number z, where w_a is a fixed non-negative real number. We perform a reduction from the minimum-cost Steiner tree problem. Notice that, for any Steiner flow x satisfying the demand constraint, we have:

$$\psi(x) = \sum_{a \in A} w_a \left(\sum_{t \in \mathcal{T}_a} x_t \right) \tag{4.19}$$

$$= \sum_{t \in \mathcal{T}} \left(\sum_{a \in t} w_a \right) x_t \tag{4.20}$$

Getting an optimal solution to this problem amounts to routing d units of flow on a minimum-cost Steiner tree with weight w_a on each arc a. Solving the minimum convex cost Steiner flow problem to optimality would hence provide at least one minimum-cost Steiner tree, while finding such a tree is an NP-hard task [43]. □

Observe that the proof above is valid in both directed and undirected networks.

4.2.5 Bounds on the value of a minimum-cost Steiner flow

Since it is difficult in all generality to solve the minimum convex cost Steiner flow problem, it seems relevant to look for computable bounds on its optimal value.

4.2.5.1 Upper bound thanks to a feasible solution

We first focus on deriving an upper bound on the optimal value ψ_S. We do this by considering the restriction of the minimum convex cost Steiner flow problem where the number of trees supporting the flow is limited to one. In this case the whole demand d has to be conveyed through a single Steiner tree. Since we are looking for a Steiner flow whose cost is minimum, it is thus natural to route the demand on a minimum-cost Steiner tree with weight $\psi_a(d)$ on each arc a as emphasized by the following lemma:

Lemma 6. *Given an instance of the minimum convex cost Steiner flow problem, a minimum-cost Steiner tree t, with weight $\psi_a(d)$ for each arc a, induces a feasible Steiner flow by routing the whole demand d on t, namely:*

$$\psi_S \leq \min_{t \in \mathcal{T}} \sum_{a \in t} \psi_a(d) \tag{4.21}$$

Finding such a minimum-cost Steiner tree is intractable, but one can always use an α-approximation algorithm in order to compute the bound in polynomial time. This bound allows one to evaluate the benefit of using more than one tree to convey flow in the network. Notice that conveying the whole demand on a single tree is likely to violate some of the network capacity requirements, which will translate in a huge penalty incurred through the cost functions. In other words, unless it is possible to satisfy the whole demand thanks to a single tree, one should expect this upper-bound to be quite loose. Since efficient mixed integer linear programming (MILP) formulations of the minimum-cost Steiner tree problem are available in the literature [55], one can also rely on a solver to find a minimum-cost Steiner tree in "affordable" time.

4.2.5.2 Lower bound on the minimum cost

To get a lower bound on the optimal value ψ_S, we will exploit the convexity of the overall cost function ψ. In the following, we denote by $\langle \cdot, \cdot \rangle$ the canonical inner product on $\mathbb{R}^{\mathcal{T}}$ and by $\nabla\psi(x)$ the gradient of function ψ, evaluated at point $x \in \mathbb{R}^{\mathcal{T}}$. We will need the following lemma:

Lemma 7. *Given a Steiner flow x, for any Steiner tree s, the partial derivative of the overall cost function ψ with respect to variable x_s, evaluated at x, can be expressed as:*

$$\frac{\partial \psi}{\partial x_s}(x) = \sum_{a \in s} \psi'_a(f_a) \tag{4.22}$$

where vector f is the arc projection of Steiner flow x.

Proof. Let f be the arc projection of the considered Steiner flow x. Through classical differential calculus we get:

$$\begin{aligned}
\frac{\partial \psi}{\partial x_s}(x) &= \frac{\partial}{\partial x_s}\left[\sum_{a \in A} \psi_a(f_a)\right] && (4.23)\\
&= \sum_{a \in A} \frac{\partial f_a}{\partial x_s} \frac{\partial \psi_a}{\partial f_a}(f_a) && (4.24)\\
&= \sum_{a \in A} \frac{\partial f_a}{\partial x_s} \psi'_a(f_a) && (4.25)\\
&= \sum_{a \in s} \psi'_a(f_a) && (4.26)
\end{aligned}$$

where the last equality comes from $f_a = \sum_{t \in \mathcal{T}_a} x_t$, which stands for any arc a. □

This lemma implies one can easily compute any component of the gradient $\nabla\psi(x)$, evaluated at point x, assuming the Steiner tree associated to this component is known. The next lemma will have important algorithmic applications:

Lemma 8. *Given any Steiner flow x, not necessarily satisfying the demand constraint, the following relation holds:*

$$\psi_S - \psi(x) \geq d \min_{t \in \mathcal{T}} \left[\sum_{a \in t} \psi'_a(f_a)\right] - \sum_{a \in A} \psi'_a(f_a) f_a \tag{4.27}$$

Proof. By convexity of ψ, we have for any two Steiner flows x and y:

$$\psi(y) - \psi(x) \geq \langle \nabla\psi(x), y - x \rangle \tag{4.28}$$

Let x^* be an optimal solution to the minimum convex cost Steiner flow problem. For any Steiner flow x, we get:

$$\psi_S - \psi(x) = \psi(x^*) - \psi(x) \tag{4.29}$$

$$\geq \langle \nabla\psi(x), x^* \rangle - \langle \nabla\psi(x), x \rangle \tag{4.30}$$

Although we do not know x^*, we can minimize the right-hand side of the previous inequality to get:

$$\psi_S - \psi(x) \geq \min_{y \in \mathcal{X}_\mathcal{T}} \langle \nabla\psi(x), y \rangle - \langle \nabla\psi(x), x \rangle \tag{4.31}$$

By Lemma 7:

$$\langle \nabla\psi(x), x \rangle = \sum_{t \in \mathcal{T}} \frac{\partial \psi}{\partial x_t}(x) x_t \tag{4.32}$$

$$= \sum_{t \in \mathcal{T}} \sum_{a \in t} \psi'_a(f_a) x_t \tag{4.33}$$

$$= \sum_{a \in A} \sum_{t \in \mathcal{T}_a} \psi'_a(f_a) x_t \tag{4.34}$$

$$= \sum_{a \in A} \psi'_a(f_a) f_a \tag{4.35}$$

We now focus on the first term in the right-hand side of Equation (4.31). By Lemma 7:

$$\min_{y \in \mathcal{X}_\mathcal{T}} \langle \nabla\psi(x), y \rangle = \min_{y \in \mathcal{X}_\mathcal{T}} \sum_{t \in \mathcal{T}} \frac{\partial \psi}{\partial x_t}(x) y_t \tag{4.36}$$

$$= \min_{y \in \mathcal{X}_\mathcal{T}} \sum_{t \in \mathcal{T}} \left(\sum_{a \in t} \psi'_a(f_a) \right) y_t \tag{4.37}$$

$$= \min_{y \in \mathcal{X}_\mathcal{T}} \sum_{a \in A} \psi'_a(f_a) \sum_{t \in \mathcal{T}_a} y_t \tag{4.38}$$

and this last minimization problem is nothing but the minimum convex cost Steiner flow problem with *linear* cost function $z \mapsto \psi'_a(f_a) z$ for each arc a. Notice that, in this problem, x is fixed, and y is the vector of variables. As previously explained (while discussing the problem complexity), an optimal solution to this last problem can be designed by routing

the whole demand d on a minimum-cost Steiner tree with weight on each arc a set to $\psi'_a(f_a)$. Hence we get:

$$\min_{y \in \mathcal{X}_\mathcal{T}} \langle \nabla \psi(x), y \rangle = d \min_{t \in \mathcal{T}} \sum_{a \in t} \psi'_a(f_a) \tag{4.39}$$

After substitution of this last result, along with the one obtained in Equation (4.35), Equation (4.31) yields the stated inequality. □

Thanks to this lemma we get the following lower bound on the optimal value ψ_S:

Lemma 9. *Given any instance of the minimum convex cost Steiner flow problem, the following inequality holds:*

$$\psi_S \geq d \min_{t \in \mathcal{T}} \sum_{a \in t} \psi'_a(0) \tag{4.40}$$

Proof. Apply Lemma 8 with the zero Steiner flow, $x \equiv 0$, whose arc projection is clearly the zero vector $f \equiv 0$. Notice that $\psi(0) = 0$ since $\psi_a(0) = 0$ for each arc a. □

Since finding a minimum-cost Steiner tree with weight $\psi'_a(0)$ on each arc a is an NP-hard task, one can instead use an approximation algorithm with approximation ratio α to find a good quality tree in polynomial time. Once this is done, simply multiply the tree cost by $\frac{d}{\alpha}$ to get a lower bound on ψ_S.

4.2.6 Algorithm

4.2.6.1 The conditional gradient method

We will now describe an algorithm to solve the minimum convex cost Steiner flow problem. Our approach rests on the *conditional gradient method*, also known as the *Frank-Wolfe algorithm* [108]. We refer the interested reader to the work of Jaggi [109] for a general presentation of this method. This algorithm has already been successfully specialized to the minimum cost multi-commodity flow problem, see the paper by Fukushima [110], the survey by Ouorou et al. [106] and references therein. Also have a look at the classical book by Bertsekas [111]. We shall now describe the proposed algorithm:

4.2.6.2 Principle of the algorithm

The algorithm proceeds in iterations which we index by i. It starts from a feasible Steiner flow $x^{(0)}$, and, at each iteration, it tries to decrease the overall cost of the current flow $x^{(i)}$ by switching flow from expensive Steiner trees to a cheaper one. A strong requirement in the design of our algorithm is that, at any iteration i, we only have access to a subset $T^{(i)}$ of *active* Steiner trees, those which were previously generated by the algorithm (see below). This means we implicitly assume that most components of the vector $x^{(i)} \in \mathbb{R}^{\mathcal{T}}$ are set to zero. Hence, great care should be taken, while updating the current solution, that the number of trees whose corresponding component in $x^{(i+1)}$ is positive remains tractable. Our algorithm ensures that at most one more tree can carry flow at each iteration, specifically $T^{(i)} \subseteq T^{(i+1)}$ and $|T^{(i+1)}| \leq |T^{(i)}| + 1$. To initialize the algorithm, find any Steiner tree and route d units of flow on it. It can be convenient to look for a minimum-cost Steiner tree with weight on each arc a set to $\psi'_a(0)$ so as to simultaneously compute the lower bound provided by Lemma 9. This gives $x^{(0)}$. At iteration i, the algorithm performs the following two steps sequentially.

4.2.6.3 First step - tree search

The first step is devoted to the search for a tree which could receive some of the flow carried by other trees, so as to decrease the overall cost of the resulting Steiner flow. A suitable candidate for such a flow transfer is a minimum-cost Steiner tree with weight $\psi'_a(f_a^{(i)})$ on each arc a, where vector $f^{(i)}$ is the arc projection of Steiner flow $x^{(i)}$. To understand why, notice that, from Lemma 7, for any Steiner tree t, the partial derivative of the overall cost function ψ with respect to variable x_t, evaluated at $x^{(i)}$, can be expressed as:

$$\frac{\partial \psi}{\partial x_t}(x^{(i)}) = \sum_{a \in t} \psi'_a(f_a^{(i)}) \tag{4.41}$$

Thus, the problem of minimizing the function $\langle \nabla\psi(x^{(i)}), x \rangle$ over $x \in \mathcal{X}_{\mathcal{T}}$ (here $\nabla\psi(x^{(i)})$ is fixed) turns out to be the minimum linear cost Steiner flow problem described above, with cost $\psi'_a(f_a^{(i)})$ for each arc a. Hence, the minimum-cost Steiner tree with respect to the aforementioned weights corresponds to the gradient descent direction. Although finding

such a tree is NP-hard, efficient mixed integer linear programming (MILP) formulations for this problem exist. Suppose we are given an oracle (say a MILP solver) which computes in "affordable" time a minimum-cost Steiner tree $t^{(i)}$. Define $T^{(i+1)} = T^{(i)} \cup \{t^{(i)}\}$. Computing $t^{(i)}$ and updating the set of active trees constitutes the first step performed by the algorithm at iteration i.

4.2.6.4 Second step - load balancing

The second step consists in updating the current flow. Assume $x^{(i)} \in \mathcal{X}_{\mathcal{T}}$ is a Steiner flow satisfying the demand constraint and using the trees in $T^{(i)}$. Let vector $f^{(i)}$ be the arc projection of Steiner flow $x^{(i)}$. We denote by $y^{(i)}$ the Steiner flow induced by routing the whole demand d on Steiner tree $t^{(i)}$. More formally, Steiner flow $y^{(i)}$ is defined as $y^{(i)}_{t^{(i)}} = d$, and $y^{(i)}_t = 0$ for any other tree $t \in T^{(i+1)} \setminus \{t^{(i)}\}$, while we *implicitly* assume $y^{(i)}_t = 0$ for any *inactive* tree $t \in \mathcal{T} \setminus T^{(i+1)}$. Steiner flow $y^{(i)}$ hence defined clearly satisfies the demand constraint. However, its cost may be far worse than $\psi(x^{(i)})$ since routing the whole demand on a single Steiner tree is likely to violate some capacity requirements, assuming the demand d is high enough. In a sense, going from $x^{(i)}$ to $y^{(i)}$ is equivalent to switching the whole flow from the trees in $T^{(i)}$ to tree $t^{(i)}$ which may be an exaggerated move. It then seems appropriate to look for a balanced decision between the two extremes of doing nothing (keeping $x^{(i)}$), and switching the whole flow on tree $t^{(i)}$ (which gives $y^{(i)}$). Therefore, it is natural to look for a convex combination $(1-\gamma)x^{(i)} + \gamma y^{(i)}$, with $\gamma \in [0,1]$, as a flow switching strategy. Let ν be the function defined on $[0,1]$ by $\nu(\gamma) = \psi([1-\gamma]x^{(i)} + \gamma y^{(i)})$. The best strategy to reduce the overall routing cost is then to pick the optimal value γ^* of the following optimization problem:

$$\min_{\gamma \in [0,1]} \nu(\gamma) = \min_{\gamma \in [0,1]} \left[\sum_{a \in t^{(i)}} \psi_a(f^{(i)}_a + \gamma(d - f^{(i)}_a)) + \sum_{a \in A \setminus t^{(i)}} \psi_a(f^{(i)}_a - \gamma f^{(i)}_a) \right] \tag{4.42}$$

Observe that, in this last problem, γ is the only variable since vectors $x^{(i)}$ and $f^{(i)}$ are fixed. The function $\gamma \mapsto \nu(\gamma)$ is convex on the interval $[0,1]$ which allows the use of a *line search* algorithm to compute γ^*. We refer the interested reader to the book by Bazaraa [112] for more details regarding the line search strategy. Define the new Steiner flow $x^{(i+1)}$

as follows:

$$x^{(i+1)} = (1 - \gamma^*)x^{(i)} + \gamma^* y^{(i)} \tag{4.43}$$

From the convexity of the feasible domain $\mathcal{X}_\mathcal{T}$, this new Steiner flow $x^{(i+1)}$ satisfies the demand constraint since the current Steiner flow $x^{(i)}$ does. Moreover we have:

$$\psi(x^{(i+1)}) = \nu(\gamma^*) \leq \nu(0) = \psi(x^{(i)}) \tag{4.44}$$

where the inequality comes from the optimality of γ^*. Equation 4.44 implies the algorithm does not deteriorate the quality of its solution through an iteration.

4.2.6.5 Convergence

By definition of tree $t^{(i)}$, Lemma 8 gives, for each iteration i:

$$\psi_S \geq \psi(x^{(i)}) + d \sum_{a \in t^{(i)}} \psi'_a(f_a^{(i)}) - \sum_{a \in A} \psi'_a(f_a^{(i)}) f_a^{(i)} \tag{4.45}$$

In the following, we denote by $(\ell^{(i)})_{i \in \mathbb{Z}_+}$ the sequence defined by:

$$\ell^{(0)} = d \min_{t \in \mathcal{T}} \sum_{a \in t} \psi'_a(0) \tag{4.46}$$

which is a lower bound on ψ_S according to Lemma 9, and, for any iteration i:

$$\ell^{(i)} = \max \left(\ell^{(i-1)}, \psi(x^{(i)}) + d \sum_{a \in t^{(i)}} \psi'_a(f_a^{(i)}) - \sum_{a \in A} \psi'_a(f_a^{(i)}) f_a^{(i)} \right) \tag{4.47}$$

For any iteration i, let $\psi^{(i)} = \psi(x^{(i)})$. From the previous discussion, it should be clear that the sequence $(\psi^{(i)})_{i \in \mathbb{Z}_+}$ is non-increasing, the sequence $(\ell^{(i)})_{i \in \mathbb{Z}_+}$ non-decreasing, and, for each iteration i, $\ell^{(i)} \leq \psi_S \leq \psi^{(i)}$. Furthermore, it can be shown that $\psi^{(i)} - \ell^{(i)} = O\left(\frac{1}{i}\right)$, by using arguments similar to those presented in [109, Theorem 2].

Recall, however, that the algorithm is polynomial in the number of calls to the oracle looking for a minimum-cost Steiner tree, but no guarantee can be made on the running time of this last subroutine.

Notice that, in the special case where all vertices except the source are receivers, the minimum convex cost spanning flow problem can be solved efficiently by applying the conditional gradient method. In this case, the sub-problem to be solved at each iteration

is to find a *minimum spanning tree* with weight $\psi'_a(f_a)$ on each arc a, where vector f is the arc projection of the current solution x of the algorithm. Such a tree can be computed in strongly polynomial time thanks to Kruskal's algorithm [49], Prim's algorithm [50], or a linear programming compact formulation [51]. When the network is directed, the Chu–Liu/Edmonds' algorithm [52, 53] returns a *minimum spanning arborescence* in strongly polynomial time.

4.2.6.6 Summary

In the following, let $\Pi_{\mathcal{T}}$ be the minimum convex cost Steiner flow problem. Given a set of Steiner trees T, denote by Π_T the restriction of problem $\Pi_{\mathcal{T}}$ to variables indexed by the trees in T. Let us summarize the behavior of the algorithm at iteration i: starting from a feasible solution to problem $\Pi_{T^{(i)}}$, the algorithm first finds a Steiner tree inducing a good descent direction, then it shifts flow from other trees to this one, according to a balance rule, in order to build a feasible solution to problem $\Pi_{T^{(i+1)}}$. The algorithm runs until a predetermined maximum number of iterations I is reached or until some fixed threshold ϵ on the gap between the value $\psi^{(i)}$ and the lower-bound $\ell^{(i)}$ is met. Figure 4.2 summarizes the whole process while Algorithm 2 below provides the corresponding pseudo-code.

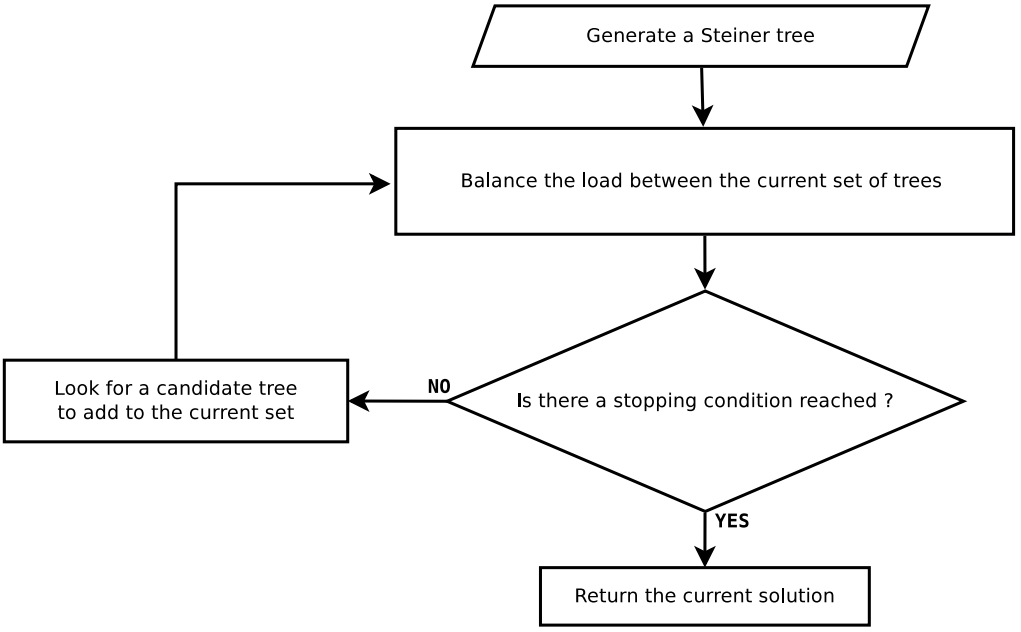


Figure 4.2 – Scheme of the proposed algorithm for solving the minimum convex cost Steiner flow problem.

Algorithm 2 : algorithm for solving the minimum convex cost Steiner flow problem.

Input: network $D = (V, A)$, source s, set of receivers R, convex cost function ψ_a for each arc a, demand d, maximum number of iterations I, threshold $\epsilon > 0$;

Output: a minimum-cost Steiner flow x;

1. Initialize x by computing a Steiner tree t_0 and routing d units of flow on it;
2. Compute the projection f of x;
3. Let $i = 0$, and $T = \{t_0\}$;
4. Compute the initial lower bound ℓ according to Equation (4.46)
5. **while** $i < I$ and $\psi(x) - \ell > \epsilon$) **do**
6. Compute $w_a = \psi'_a(f_a)$ for each arc a;
7. Find a minimum-cost Steiner tree t with respect to weights w thanks to a subroutine;
8. Update the set of active trees, $T = T \cup \{t\}$;
9. Use line-search to find the optimal solution γ^* to Problem (4.42);
10. Update the current flow x and its projection f according to Equation (4.43);
11. Update the lower bound ℓ according to Equation (4.47);
12. $i = i + 1$;
13. **end while**
14. Return x, f, and T.

4.2.6.7 Improvement

The proposed algorithm can be improved by using the so-called *gradient projection method* [113] which has already been successfully specialized to the minimum cost multi-commodity flow problem, an approach referred to as the *flow deviation method* in the literature, see [114, 115, 111, 106].

4.3 Minimum convex cost coded flow

4.3.1 Problem statement

We now turn our attention to the problem of finding a coded flow with minimum cost. As in the previous chapters, we will make a distinction between directed and bidirected networks on the one hand, and undirected networks on the other hand. To summarize the distinction, the expression of the cost of a coded flow depends on the network structure. The setting is exactly the same as in the previous problem: the instance is made of a *network* $D = (V, A)$ with *source* s, set R of *receivers*, and a cost function ψ_a for each arc a, along with a positive real *demand* d. We also keep all our previous assumptions on the family of functions $(\psi_a)_{a \in A}$. Hence, we will also relax all capacity requirements, assuming again that, for each arc a such that $c_a \leq d$, the penalty $\psi_a(f_a)$ incurred by routing a flow of value $f_a \in [c_a, d]$ is high enough to prevent any capacity violation. Recall that a *coded flow* x is an assignment of a non-negative real value x_p to each path p between s and a receiver r. We denote by $\mathcal{P}^r$ the set of such paths, by $\mathcal{P}$ the union of those sets $\mathcal{P}^r$ for all $r \in R$, and by $\mathcal{P}^r_a$ the subset of $\mathcal{P}^r$ using arc a. Given a coded flow x, we call x_p the *flow* on path p, f^r_a the *flow induced on arc a toward receiver r*, namely $f^r_a = \sum_{p \in \mathcal{P}^r_a} x_p$, and h_a the *coded flow* on arc a, i.e. $h_a = \max_{r \in R} f^r_a$ (notice the use of the max operator instead of the sum). We again refer to vector f and h respectively as the *arc projection* and the *coding vector* associated to coded flow x. Given a capacity vector c on the arcs, a coded flow x satisfies the capacity requirements if, for each arc a, the amount of *coded flow* on a is smaller than, or equal to, the arc capacity, namely $h_a \leq c_a$. Furthermore, we say that x meets the demand if each receiver r experiences a throughput of value d, i.e. $\sum_{p \in \mathcal{P}^r} x_p = d$. We define the overall cost of a coded flow x as in Equation (4.1), except

that the coding vector h is now playing the role previously fulfilled by the arc projection f, which yields:

$$\psi(x) = \sum_{a \in A} \psi_a(h_a) \tag{4.48}$$

Given a coded flow x and its coding vector h, along with an arc a, we call $\psi_a(h_a)$ the *cost induced by x on arc a*, while the *overall cost of x* is the quantity $\psi(x)$. Notice that this overall cost is again separable with respect to the arcs but not with respect to the paths. In the following, we denote by $\mathcal{X}_\mathcal{P}$ the set of all coded flows meeting the demand constraints:

$$\mathcal{X}_\mathcal{P} = \left\{ x \in \mathbb{R}_+^\mathcal{P} : \sum_{p \in \mathcal{P}^r} x_p = d \, \forall r \in R \right\} \tag{4.49}$$

A coded flow x is *feasible* if $x \in \mathcal{X}_\mathcal{P}$. We are now ready to formally express this new problem:

Problem	*Minimum convex cost (uncapacitated) coded flow*
Instance	Network $D = (V, A)$, source s, set of receivers R, convex cost function ψ_a for each arc a, positive demand d
Solution	A feasible coded flow $x \in \mathcal{X}_\mathcal{P}$ in D
Objective	Minimize the overall cost $\psi(x)$ of coded flow x as defined in Equation (4.48)

We denote by ψ_C the value of a minimum-cost coded flow for the considered instance:

$$\psi_C = \min_{x \in \mathcal{X}_\mathcal{P}} \psi(x) \tag{4.50}$$

Notice that, when the set R is a singleton, the minimum convex cost coded flow problem is exactly the minimum convex cost flow problem between the source and the unique receiver.

4.3.2 Example continued

Consider again the directed butterfly network depicted on Figure 4.1 a). This time, we are studying the minimum cost coded flow problem under the same setting as in the previous example. Namely, the demand is set to $d = 1$, and the cost function ψ_a on each arc a is $\psi_a(z) = \frac{z}{c_a - z}$ for any $z \in [0, d]$ where the capacity of each arc a is $c_a = 1$. A minimum-cost coded flow $\varkappa$ for this instance is depicted on Figure 4.3. The coded flow $\varkappa$ uses two paths to convey the data from the source to each receiver. The overall cost induced by coded flow $\varkappa$ is 8.94.

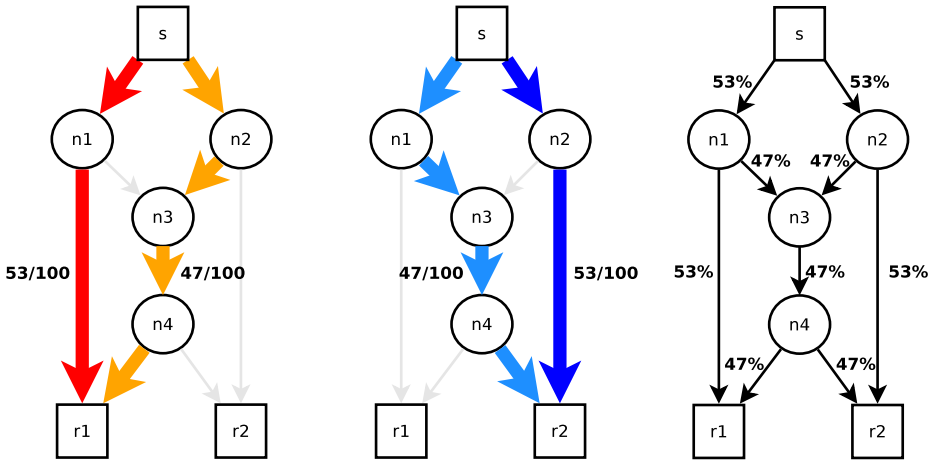


Figure 4.3 – A minimum-cost coded flow χ. It uses exactly two paths to route the data from the source to each receiver. Each path using the arc (n_3, n_4) conveys 47/100 units of flow, while each of the two other paths routes 53/100 units of flow. The third network on the right provides the percentage of the load $\frac{f_a}{c_a}$ for each arc a. The overall cost induced by the coded flow χ is 8.94.

4.3.3 Mathematical programming formulations

4.3.3.1 Directed and bidirected networks

The minimum convex cost coded flow problem can also be formalized using mathematical programming:

$$\begin{cases} \psi_C = \min \sum_{a \in A} \psi_a(h_a) & (4.51) \\ \text{s.t.} \quad \sum_{p \in \mathcal{P}^r} x_p = d \quad \forall r \in R & (4.52) \\ h_a \geq \sum_{p \in \mathcal{P}^r_a} x_p \quad \forall a \in A, r \in R & (4.53) \\ h_a, x_p \geq 0 \quad \forall a \in A, p \in \mathcal{P} & (4.54) \end{cases}$$

For each path p, variable x_p stands for the amount of flow routed on p, while, for each arc a, variable h_a models the amount of flow going through a. Observe that, since we may have an exponential number of paths in $\mathcal{P}$, the previous model may also have an exponential number of variables. Constraints (4.52) ensure that, for each receiver, the coded flow x meets the demand d. Combined with the assumption that each function ψ_a is non-decreasing and the fact that the objective function is to be minimized, Constraints (4.53) enforce variable h_a to take the value $\max_{r \in R} \sum_{p \in \mathcal{P}^r_a} x_p$, for any arc a, in an optimal solution.

Although this path formulation may involve a huge number of variables, the following arc formulation of the minimum convex cost coded flow problem was presented by Lun et al. in [116]:

$$\begin{cases} \psi_C = \min \sum_{a \in A} \psi_a(h_a) & (4.55) \\ \text{s.t.} \sum_{a \in \delta^+(v)} f_a^r - \sum_{a \in \delta^-(v)} f_a^r = b_v^r(d) \quad \forall r \in R, v \in V & (4.56) \\ h_a \geq f_a^r \quad \forall a \in A, r \in R & (4.57) \\ f_a^r, h_a \geq 0 \quad \forall a \in A, r \in R & (4.58) \end{cases}$$

with

$$b_v^r(d) = \begin{cases} d & \text{if } v = s \quad (4.59) \\ -d & \text{if } v = r \quad (4.60) \\ 0 & \text{otherwise} \quad (4.61) \end{cases}$$

For each arc a and each receiver r, variable f_a^r stands for the amount of flow from source s to r going through a, while variable h_a models the amount of coded flow conveyed through a. Observe that, again, we implicitly use the monotonicity of the cost functions in the definition of Constraints (4.57) so as to get $h_a = \max_{r \in R} f_a^r$, for any arc a, in an optimal solution.

4.3.3.2 Undirected networks

To understand why the model of the minimum convex cost coded flow problem in an undirected network differs from the one in a directed network, consider an undirected network $G = (V, E)$ along with the bidirected network $B = (V, L)$ obtained from G by replacing each edge e by a *pair of directed arcs*, or *link* while keeping the source and the receivers as defined in G. The difference between the two models lies in that, in the undirected framework, there is a cost function ψ_ℓ *associated to each link* ℓ of the bidirected network (ψ_ℓ is actually the cost function ψ_e of the edge e from which link ℓ originates) while in the bidirected setting, *each arc of the pair is equipped with its own cost function.* Observe that the two models remain distinct even if we assume that both cost functions are equal, namely $\psi_{a_1} = \psi_{a_2}$ (denoted by ψ_ℓ in the following) for each link $\ell = \{a_1, a_2\}$ in bidirected network B: Given a coded flow x in bidirected network B along with its

coding vector h, all that can be say is that $\psi_\ell(h_{a_1}) + \psi_\ell(h_{a_2})$ has no reason to be equal to $\psi_\ell(h_{a_1} + h_{a_2})$. As already pointed out by Yin et al. [42] the two models become equivalent when each cost function of the network is *linear*.

To compute a minimum convex cost coded flow in an undirected network $G = (V, E)$, solve the problem obtained by replacing in the bidirected network $B = (V, L)$ associated to G the objective function by the following one:

$$\sum_{\ell \in L} \psi_\ell(h_{a_1} + h_{a_2}) \tag{4.62}$$

where, for each link $\ell = \{a_1, a_2\}$, the cost function ψ_ℓ is set to the cost function ψ_e of the edge e from which link ℓ originates.

4.3.4 Complexity and algorithms

Regardless of the network structure, the arc formulation of the minimum convex cost coded flow problem can be regarded as a problem of minimizing a convex function over a convex set induced by a *polynomial number* of linear constraints. Therefore, we have all standard techniques of convex optimization at our disposal [105] to compute an arc decomposition of a minimum-cost coded flow in polynomial time. Given such an arc decomposition f, a path decomposition x can be built in polynomial time thanks to the technique presented in the proof of Lemma 1: For each receiver r, let f^r be the vector of components f^r_a over each arc a of the network. The vector f^r is the arc projection of a *simple* flow between the source and receiver r. For each receiver r, compute a path decomposition of this simple flow using the algorithm presented by Ahuja et al. [7, Theorem 3.5] which runs in polynomial time. The concatenation of all those path decompositions (one per receiver) is clearly a valid path decomposition of the coded flow.

A decentralized algorithm to solve the minimum convex cost coded flow problem is proposed by Lun et al. in [116]. If one is interested in the game theoretic approach where each receiver tries to selfishly minimize its own cost, the proof of existence of a local Nash equilibrium along with an algorithm to find one are presented by Bhadra et al. in [117]. Notice that, considering the arc formulation of the minimum convex cost coded flow problem, we could directly incorporate the capacity requirements to the set of constraints,

instead of assuming them implicitly through the cost functions, while keeping a tractable problem. We chose to relax those capacity requirements to allow comparisons between information flows.

4.3.5 A new algorithm

4.3.5.1 The conditional gradient method

We will now propose an algorithm to solve the minimum convex cost coded flow problem. We simply pick the conditional gradient method as with the previous problem. Therefore, this algorithm is very similar to Algorithm 2. The main differences arise from the selection of the initial coded flow $x^{(0)}$ along with the linear subproblem to be solved at each iteration. Notice however that the function $x \mapsto \sum_{a \in A} \psi_a \left(\max_{r \in R} \left[\sum_{p \in \mathcal{P}_a^r} x_p \right] \right)$ from $\mathcal{X}_{\mathcal{P}}$ to $\mathbb{R}_+$ is *not* differentiable with respect to variable x_p, for any path p between the source and a receiver.

In the following we will specialize the conditional gradient method for solving the arc formulation of the minimum convex cost coded flow problem. We denote by $\mathcal{F}$ the polytope induced by Constraints (4.56)-(4.57)-(4.58). Given a vector $(f, h) \in \mathcal{F}$, we write $\psi(f, h) = \sum_{a \in A} \psi_a(h_a)$. The mapping $(f, h) \mapsto \psi(f, h)$ is clearly differentiable on $\mathcal{F}$ and for each arc a:

$$\frac{\partial \psi}{\partial h_a}(f, h) = \psi_a'(h_a) \tag{4.63}$$

while clearly, for each arc a and each receiver r:

$$\frac{\partial \psi}{\partial f_a^r}(f, h) = 0 \tag{4.64}$$

so that we can easily compute the gradient $\nabla \psi(f, h)$ of the mapping $(f, h) \mapsto \psi(f, h)$, evaluated at point (f, h). Thus, we can find a good descent direction at a given point (f, h), by solving the following problem:

$$\min_{(\mathbb{f}, \mathbb{h}) \in \mathcal{F}} \langle \nabla \psi(f, h), (\mathbb{f}, \mathbb{h}) \rangle = \min_{(\mathbb{f}, \mathbb{h}) \in \mathcal{F}} \sum_{a \in A} \psi_a'(h_a) \mathbb{h}_a \tag{4.65}$$

Observe that the sub-problem defined in Equation (4.65) is exactly the minimum convex cost coded flow problem with *linear* cost function $z \mapsto \psi_a'(h_a) z$ set on each arc a. Fur-

thermore, solving this sub-problem provides a lower bound on ψ_C, as emphasized by the following lemma:

Lemma 10. *Given any vector $(f,h) \in \mathbb{R}_+^{A\times R} \times \mathbb{R}_+^A$, the following relation holds:*

$$\psi_C - \psi(f,h) \geq \min_{(\mathbb{f},\mathbb{h})\in\mathcal{F}} \sum_{a\in A} \psi'_a(h_a)\mathbb{h}_a - \sum_{a\in A} \psi'_a(h_a)h_a \tag{4.66}$$

Proof. The argument is the same as the one used in the proof of Lemma 8. By convexity of the mapping $(f,h) \mapsto \psi(f,h)$, we have for any two vectors (f,h) and $(\mathbb{f},\mathbb{h})$ of $\mathbb{R}_+^{A\times R} \times \mathbb{R}_+^A$:

$$\psi(f,h) - \psi(\mathbb{f},\mathbb{h}) \geq \langle \nabla\psi(f,h), (\mathbb{f},\mathbb{h}) - (f,h)\rangle \tag{4.67}$$

from which we deduce:

$$\psi_C - \psi(f,h) \geq \min_{(\mathbb{f},\mathbb{h})\in\mathcal{F}} \langle \nabla\psi(f,h), (\mathbb{f},\mathbb{h})\rangle - \langle \nabla\psi(f,h), (f,h)\rangle \tag{4.68}$$

From Equations (4.63)-(4.64), we have:

$$\langle \nabla\psi(f,h), (f,h)\rangle = \sum_{a\in A} \psi'_a(h_a)h_a \tag{4.69}$$

Substituting this last result, along with the one obtained in Equation (4.65), Equation (10) yields the stated inequality. □

Thanks to Lemma 10 we get the following lower bound on the optimal value ψ_C:

Lemma 11. *Given any instance of the minimum convex cost coded flow problem, the following inequality holds:*

$$\psi_C \geq \min_{x\in\mathcal{X}_\mathcal{P}} \sum_{a\in A} \psi'_a(0)h_a \tag{4.70}$$

where $h_a = \max_{r\in R}\left[\sum_{p\in\mathcal{P}_a^r} x_p\right]$ for each arc a.

Proof. Notice that $\psi(0,0) = 0$ since $\psi_a(0) = 0$ for each arc a. By applying Lemma 10 with the zero vector, $(f,h) \equiv (0,0)$, one obtains:

$$\psi_C \geq \min_{(\mathbb{f},\mathbb{h})\in\mathcal{F}} \sum_{a\in A} \psi'_a(0)\mathbb{h}_a = \min_{x\in\mathcal{X}_\mathcal{P}} \sum_{a\in A} \psi'_a(0)h_a \tag{4.71}$$

where in the last equality $h_a = \max_{r\in R}\left[\sum_{p\in\mathcal{P}_a^r} x_p\right]$ for each arc a. □

The problem defined by the right-hand side of Equation (4.70) could simultaneously provide a feasible coded flow $x^{(0)}$ along with an initial lower-bound $\ell^{(0)}$ on the optimal value ψ_C, assuming it is practically solvable. Hence, it turns out that an efficient subroutine to solve the minimum *linear* cost coded flow problem is the only tool required to apply the conditional gradient method to the original problem. Fortunately, the arc formulation of this sub-problem can be solved to optimality in polynomial time by linear programming.

4.3.5.2 Principle of the algorithm

The algorithm initialization consists in computing the lower bound on ψ_C presented in Lemma 11. This initialization step also provides a feasible vector $(f^{(0)}, h^{(0)}) \in \mathcal{F}$. The algorithm then proceeds in iterations which we index by i. At any iteration i, the algorithm tries to improve its current solution $(f^{(i)}, h^{(i)})$ by performing the following two steps sequentially. The first step is devoted to the search for a candidate vector $(\mathbb{f}^{(i)}, \mathbb{h}^{(i)}) \in \mathcal{F}$, on which some flow could be transferred so as to reduce the overall cost of the designed solution. To do this, the algorithm calls a subroutine which returns an optimal solution $(\mathbb{f}^{(i)}, \mathbb{h}^{(i)})$ to the arc formulation of the minimum *linear* cost coded flow problem for the instance where the cost of each arc a is $\psi'_a(h_a^{(i)})$. Computing the vector $(\mathbb{f}^{(i)}, \mathbb{h}^{(i)})$ constitutes the first step performed by the algorithm at iteration i. The second step consists in updating the current solution by finding a balance between vectors $(f^{(i)}, h^{(i)})$ and $(\mathbb{f}^{(i)}, \mathbb{h}^{(i)})$. Let ν be the function defined on $[0, 1]$ by $\nu(\gamma) = \psi([1 - \gamma](f^{(i)}, h^{(i)}) + \gamma(\mathbb{f}^{(i)}, \mathbb{h}^{(i)}))$. The best strategy to reduce the overall routing cost is then to pick the optimal value γ^* of the following optimization problem:

$$\min_{\gamma \in [0,1]} \nu(\gamma) \tag{4.72}$$

Observe that, in this last problem, γ is the only variable since both vectors $(f^{(i)}, h^{(i)})$ and $(\mathbb{f}^{(i)}, \mathbb{h}^{(i)})$ are fixed. The function $\gamma \mapsto \nu(\gamma)$ is convex on the interval $[0, 1]$ allowing the use of a *line search* algorithm to find γ^*, see [112]. The set $\mathcal{F}$ is clearly convex ($\mathcal{F}$ is a polytope). Thus, the algorithm can update the current solution according to the following rule:

$$(f^{(i+1)}, h^{(i+1)}) = (1 - \gamma^*)(f^{(i)}, h^{(i)}) + \gamma^*(\mathbb{f}^{(i)}, \mathbb{h}^{(i)}) \tag{4.73}$$

In order to complete this second step, it remains to compute the following lower-bound on ψ_C which is obtained by combining Lemma 10 with the definition of vector $(f^{(i)}, h^{(i)})$ as follows:

$$\ell^{(i)} = \max\left(\ell^{(i-1)}, \psi(f^{(i)}, h^{(i)}) + \sum_{a \in A} \psi'_a(h_a^{(i)})[\mathbb{h}_a^{(i)} - h_a^{(i)}]\right) \tag{4.74}$$

4.3.5.3 Summary

For any non-negative integer i, let $\psi^{(i)} = \psi(f^{(i)}, h^{(i)})$. The algorithm runs until a predetermined maximum number of iterations I is reached or until some fixed threshold ϵ on the gap between the value $\psi^{(i)}$ and the lower bound $\ell^{(i)}$ is met. By using arguments similar to those presented in [109, Theorem 2], it can be shown that $\psi^{(i)} - \ell^{(i)} = O\left(\frac{1}{i}\right)$. At each iteration the algorithm calls a subroutine to solve a linear programming problem involving a polynomial number of continuous variables and constraints. Therefore, the overall procedure runs in polynomial time. Algorithm 3 provides the corresponding pseudo-code.

Algorithm 3 : algorithm for solving the minimum convex cost coded flow problem.

Input: network $D = (V, A)$, source s, set of receivers R, convex cost function ψ_a for each arc a, demand d, maximum number of iterations I, threshold $\epsilon > 0$;
Output: a minimum-cost coded flow (f, h);

1. Initialize vector (f, h) by calling the subroutine on the linear problem introduced in Lemma 11;
2. Initialize lower-bound ℓ with the optimal value of this previous problem;
3. Let $i = 0$;
4. **while** $i < I$ and $\psi(f, h) - \ell > \epsilon$) **do**
5. Compute $w_a = \psi'_a(h_a)$ for each arc a;
6. Call the subroutine to get an optimal solution $(\mathbb{f}, \mathbb{h})$ to the minimum linear cost coded flow problem with weight w_a on each arc a;
7. Use line-search to compute an optimal solution γ^* to Problem (4.72);
8. Update the current vector (f, h) according to Equation (4.73);
9. Update the lower bound ℓ according to Equation (4.74);
10. $i = i + 1$;
11. **end while**
12. Return the couple of vectors (f, h).

The reader may have notice that the algorithm, as defined above, does not return a path decomposition of the coded flow, while such a decomposition is often desirable

in telecommunication applications. To circumvent this issue, recall that given a feasible solution (f, h) for the arc formulation, a feasible solution (x, h) for the path formulation of same value as (f, h) can be built in polynomial time thanks to the technique presented in the proof of Lemma 1.

4.3.5.4 Improvement

It would be possible to solve the minimum convex cost coded flow problem by using the previously mentioned *gradient projection method* [113], also known as the *flow deviation method* in the literature [114, 115, 111, 106].

4.3.6 Coding scheme

We assume that there is at least one feasible coded flow satisfying all requirements of the considered instance. This assumption can be tested in polynomial time by computing a maximum coded flow in the instance of interest, as explained in Section 2.3, and by comparing its value φ_C with the demand d. We shall further assume that the coded flow x returned by Algorithm 3 satisfies all practical capacity requirements, namely $h_a \leq c_a$ for any arc a, where h is the coding vector of coded flow x. This can be done without loss of generality since, as previously hinted, one can always add those capacity requirements as explicit constraints in the model, without altering its complexity. Furthermore, the conditional gradient method is also left unmodified by this evolution of the model. Thus, any of the methods intended to design a coding scheme which have been previously mentioned in Section 2.3 can be applied to find a suitable coding scheme, ensuring a rate of value d.

4.4 Features of the information flows

4.4.1 Cost coding gain

In this section, we are interested in the following question:

Question 3. *Is there an incentive, with respect to the overall routing cost, to use network coding in a multicast network?*

To facilitate comparisons, we introduce the following notion:

Definition 4. *Given a network $D = (V, A)$ with source s, set of receivers R, cost function ψ_a for each arc a, along with positive demand d, the* cost coding gain g_ψ*, provided by network coding over multicast alone, is the ratio of the value of a minimum-cost Steiner flow over the one of a minimum-cost coded flow, namely:*

$$g_\psi = \frac{\psi_S}{\psi_C} \tag{4.75}$$

Recall that we only consider networks where there is at least one path between the source and each receiver. Once combined with the absence of explicit capacity requirements, it should be clear that this last property implies that both minimum-cost information flow problems are feasible. We shall further assume that the quantity ψ_C is positive so that the previous definition of the *cost coding gain* g_ψ is well-grounded. The next theorem shows that adding the possibility to perform coding operations in a multicast network never impedes the cost one has to pay to use this network:

Theorem 18. *Given a network $D = (V, A)$ with source s, set of receivers R, cost function ψ_a for each arc a, along with positive demand d, the cost coding gain g_ψ is at least* 1*.*

The proof of Theorem 18 is very similar to the one of Theorem 3 and can be found in Appendix C.1.3. Notice that, with respect to the overall cost, there is an incentive to use network coding only if $g_\chi > 1$. We shall now study the cost coding gain of some networks.

4.4.2 Features of the cost coding gain

4.4.2.1 Cost coding gain in the single receiver case

We first give a word about the setting where there is only one single receiver. In this case both the minimum convex cost Steiner flow problem and the minimum convex cost coded flow problem become equivalent to the minimum convex cost flow problem, as emphasized by the following lemma:

Lemma 12. *In a network $D = (V, A)$ with source s, a single receiver r distinct from s, cost function ψ_a for each arc a, along with positive demand d, there is no incentive to use either multicast or network coding and $g_\psi = 1$.*

4.4.2.2 Linear and nonlinear cost coding gain

The following theorem provides a connection between the linear cost coding gain and the nonlinear one.

Theorem 19. *Given a network $D = (V, A)$ with source s, set of receivers R, convex cost function ψ_a for each arc a, along with positive demand d, the associated cost coding gain g_ψ is upper-bounded by $\beta_\psi(d)g_\Psi$, where g_Ψ is the cost coding gain associated to the instance where the cost function on each arc a is $z \mapsto \psi'_a(0)z$, and $\beta_\psi(d) = \max_{a\in A} \frac{\psi_a(d)}{d\psi'_a(0)}$.*

Proof. Recall from Equation (4.70) that ψ_C is lower-bounded by the overall cost Ψ_C of a minimum-cost coded flow, satisfying demand d, with *linear* cost function $z \mapsto \psi'_a(0)z$ set on each arc a, namely $\Psi_C \leq \psi_C$. Now, let $\mathbb{t}$ be a minimum-cost Steiner tree with respect to weight $\psi'_a(0)$ on each arc a. Following our discussion about Lemma 6, it should be clear that

$$\psi_S \leq \sum_{a\in\mathbb{t}} \psi_a(d) \tag{4.76}$$

Furthermore, by definition of $\beta_\psi(d)$, we have:

$$\sum_{a\in\mathbb{t}} \psi_a(d) \leq \beta_\psi(d)d \sum_{a\in\mathbb{t}} \psi'_a(0) \tag{4.77}$$

Now, observe that the Steiner flow defined by routing the whole demand d on Steiner tree $\mathbb{t}$ is an optimal solution to the minimum cost Steiner flow problem where the linear cost function on each arc a is $z \mapsto \psi'_a(0)z$. This directly results from the definition of Steiner tree $\mathbb{t}$ and the fact that routing the whole demand d on a single tree is an optimal routing strategy when the cost function on each arc is linear. Let Ψ_S be the value of this minimum-cost Steiner flow, namely $\Psi_S = d\sum_{a\in\mathbb{t}} \psi'_a(0)$. Combining Equation (4.76) with Equation (4.77) and the definition of Ψ_S leads to:

$$\psi_S \leq \beta_\psi(d)\Psi_S \tag{4.78}$$

Introducing the *linear* cost coding gain $g_\Psi = \frac{\Psi_S}{\Psi_C}$ into Equation (4.78) and using the inequality $\Psi_C \leq \psi_C$ yields:

$$\psi_S \leq \beta_\psi(d)g_\Psi\psi_C \tag{4.79}$$

or equivalently:

$$g_\psi \leq \beta_\psi(d) g_\Psi \tag{4.80}$$

which is the desired inequality. □

We now give a word about the quantity $\beta_\psi(d)$. For any arc a and any $z \in [0, d]$, the convexity of the cost function ψ_a implies:

$$\psi_a(z) \geq \psi_a(0) + \psi'_a(0)z = \psi'_a(0)z \tag{4.81}$$

where the equality comes from the assumption $\psi_a(0) = 0$. From Equation (4.81) we have $\beta_\psi(d) \geq 1$, provided the cost function ψ_a is convex for each arc a. Furthermore:

Lemma 13. *$\beta_\psi(d) = 1$ if and only if, for each arc a, the cost function ψ_a is linear.*

Proof. If ψ_a is linear for each arc a then $\psi_a(d) = \psi'_a(0)d$ and $\beta_\psi(d) = 1$. Conversely, if $\beta_\psi(d) = 1$ then, for each arc a, $\psi_a(d) = \psi'_a(0)d$ thanks to Equation (4.81). Hence, for each arc a and any $z \in [0, d]$, we have by convexity of ψ_a:

$$\psi_a(z) = \psi_a\left(\frac{z}{d}d + \left(1 - \frac{z}{d}\right)0\right) \leq \frac{z}{d}\psi_a(d) + \left(1 - \frac{z}{d}\right)\psi_a(0) = \psi'_a(0)z \tag{4.82}$$

where the second equality comes from $\psi_a(0) = 0$. Combining Equation (4.81) with Equation (4.82), we get $\psi_a(z) = \psi'_a(0)z$ for any $z \in [0, d]$ and ψ_a is linear for each arc a. □

It should be clear by now that $\beta_\psi(d)$ is a measure of the non-linearity of the overall cost function ψ. Thus, Theorem 19 implies that the more nonlinear this overall cost function ψ is, the wider the cost coding gain domain $[1, \beta_\psi(d)g_\Psi]$ will be. The values of $\beta_\psi(d)$ for some convex cost functions are given in Table 4.1.

Type	Formula	Domain	$\beta_\psi(d)$
Linear	$z \mapsto w_1 z$	$\mathbb{R}_+$	1
Square	$z \mapsto w_2 z^2$	$\mathbb{R}_+$	$+\infty$
Quadratic	$z \mapsto w_2 z^2 + w_1 z$	$\mathbb{R}_+$	$1 + \frac{w_2}{w_1}d$
Exponential	$z \mapsto \exp(z) - 1$	$\mathbb{R}_+$	$d^{-1}(\exp(d) - 1)$
Kleinrock	$z \mapsto \frac{z}{1-z}$	$[0, 1[$	$(1 - d)^{-1}$

Table 4.1 – Value of $\beta_\psi(d)$ for some convex cost functions ψ.

4.4.2.3 Coding gain and linear cost coding gain

An important result regarding the cost coding gain in networks whose channels are endowed with *linear* cost functions is given by Yin et al. [79]. Their result can be stated as follows:

Theorem 20. *[79, Theorem 9] Given a network $D = (V, A)$ with source s and set of receivers R, denote by $g_\varphi(D, c)$ the* coding gain *obtained by setting capacity c_a on each arc a. Also denote by $g_\psi(D, w)$ the* cost coding gain *obtained by setting* linear *cost function $z \mapsto w_a z$ on each arc a along with positive demand d. Then,*

$$\max_{c \geq 0} g_\varphi(D, c) = \max_{w \geq 0} g_\psi(D, w) \tag{4.83}$$

This last result is fundamental in that it establishes a link between two a priori unrelated quantities, namely the coding gain g_φ and the linear cost coding gain g_ψ. Beware that, in [79], this last theorem is stated for any *hyper-network* which encompasses the case of all classical networks.

4.4.2.4 Bidirected and undirected networks

The next theorem, due to Yin et al. [42, 79], gives an upper bound on the *linear* cost coding gain of any undirected network:

Theorem 21. *[42, Theorem 6] Given an undirected network $G = (V, E)$ with source s, set of receivers R, linear cost function $z \mapsto w_e z$ on each edge e, along with positive demand d, the associated cost coding gain g_ψ is upper-bounded by 2.*

Theorem 21 comes from Theorem 20 combined with the upper bound of 2 on the coding gain g_φ of any undirected network [81]. The result below immediately follows from Theorem 21 combined with Theorem 19:

Corollary 8. *Given an undirected network $G = (V, E)$ with source s, set of receivers R, convex cost function ψ_e on each edge e, along with positive demand d, the associated cost coding gain g_ψ is upper-bounded by $2\beta_\psi(d)$, where $\beta_\psi(d) = \max_{e \in E} \frac{\psi_e(d)}{d\psi'_e(0)}$.*

Notice that, this last upper bound on the cost coding gain of an undirected network is function of $\beta_\psi(d)$. From Lemma 13, when $\beta_\psi(d) = 1$ we get back Theorem 21.

Recall that we restrict our attention to bidirected networks where the cost functions of two reverse arcs are equal, $\psi_{a_1} = \psi_{a_2}$ for each link $\ell = \{a_1, a_2\}$. Exploiting the equivalence between an undirected *uncapacitated* network and a bidirected one, we immediately deduce that the upper bound of $2\beta_\psi(d)$ also holds for bidirected networks by setting $\beta_\psi(d) = \max_{\ell \in L} \frac{\psi_\ell(d)}{d\psi'_\ell(0)}$ where ψ_ℓ is the cost function associated to link ℓ.

4.4.2.5 Directed networks

The following result is mentioned by Maheshwar et al. [85] while discussing a conjecture:

Theorem 22. *[85, regarding Conjecture 8.1] For any real number $\eta \geq 1$, there exists an infinite family of directed networks with a cost function on each arc which is either the identity function $z \mapsto z$ or the zero function $z \mapsto 0$, such that the ratio between the value of a minimum-cost Steiner flow and the one of a minimum-cost coded flow satisfies $g_\psi \geq \eta$.*

Thus, the cost coding gain can be arbitrarily large in directed networks, even if the cost function on each arc is restricted to be either the identity function $z \mapsto z$ or the zero function $z \mapsto 0$. Again, the proof rests on the family of directed combination networks introduced in [83, 84]. Interestingly, the analysis conducted in [85] also leads to the following result:

Theorem 23. *[85, Theorem 4.1] In a directed combination network with the same linear cost function $z \mapsto wz$ on each arc, the associated cost coding gain g_ψ is tightly upper-bounded by $\frac{9}{8}$.*

Beware that [85] primarily deals with *undirected combination networks*, yet the proof can be straightforwardly adapted to directed ones. This last theorem implies that, although one can expect an arbitrarily large throughput gain in a directed combination network, enforcing uniform linear cost on the arcs may prevent any substantial cost gain. We shall slightly extend this last result as follows:

Corollary 9. *In a directed combination network with given demand d, and the same convex cost function ψ on each arc, the associated cost coding gain g_ψ is tightly upper-bounded by $\frac{9}{8}\beta_\psi(d)$, where $\beta_\psi(d) = \frac{\psi(d)}{d\psi'(0)}$.*

Proof. Combine Theorem 23 with Theorem 19 to get the desired inequality. □

From Theorem 22 and Corollary 9, it appears that, as far as directed combination networks are concerned, having *different linear* cost functions on the arcs may allow a greater benefit from network coding techniques than *uniform nonlinear* convex cost functions.

4.4.3 Experimental evaluation of the convex cost coding gain

4.4.3.1 Setting

We use the same fifteen network topologies taken from the SNDlib library [97] as in the previous chapter. We then arbitrarily pick a node as source, and a subset of nodes as receivers. The capacity of each channel is picked in the set [10] (of all integers between 1 and 10) uniformly at random. The demand is chosen sufficiently low so as to allow the existence of a feasible Steiner flow with respect to capacity requirements. Each channel is endowed with a Kleinrock-like convex cost function which is parametrized by the channel capacity. More formally, let κ be the function from $[0,1[$ to $\mathbb{R}_+$ defined by $\kappa(z) = \frac{z}{1-z}$ and denote by κ' the first derivative of κ. Also let ω be a small positive real number. For any arc a, we set the cost function ψ_a as follows:

$$\psi_a(z) = \begin{cases} \kappa\left(\dfrac{z}{c_a}\right) & \text{if } z \in [0, (1-\omega)c_a] \quad (4.84) \\ \kappa'(1-\omega)[z - c_a] + \kappa(1-\omega) & \text{if } z \in [(1-\omega)c_a, d] \quad (4.85) \end{cases}$$

where c_a is the capacity of arc a. Hence, each function ψ_a is nonlinear on $[0, (1-\omega)c_a]$, and linear on $[(1-\omega)c_a, d]$ (assuming this last set is not empty). Notice that the function ψ_a and its derivative ψ'_a are continuous on $[0, d]$, and satisfy all the requirements listed in the beginning of this chapter, provided the value of ω is small enough so as to penalize heavily any capacity violation, say $\omega = 10^{-2}$ in most practical applications. The shape of the resulting cost function ψ_a is depicted in Figure 4.4.

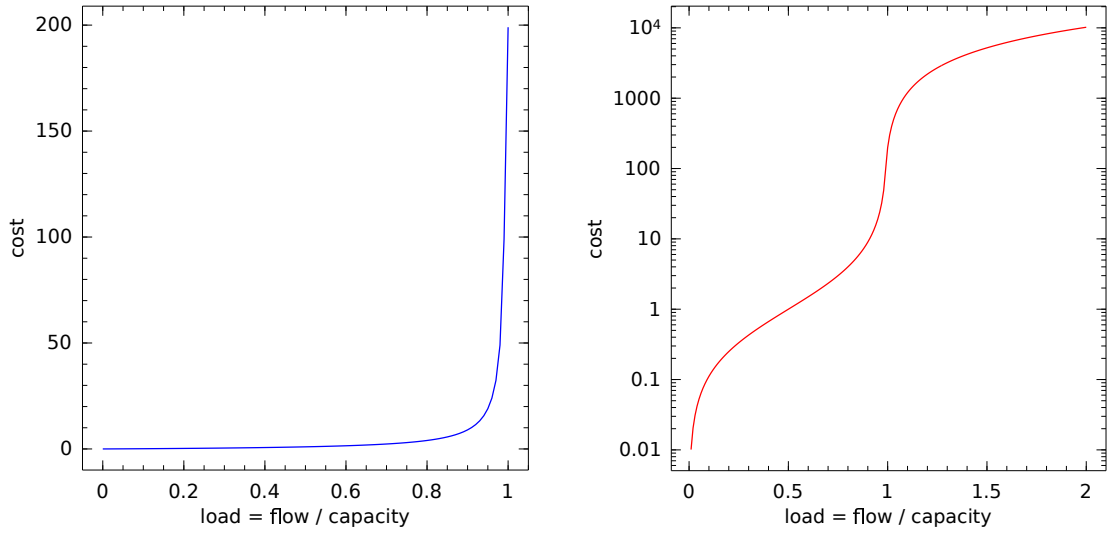


Figure 4.4 – Shape of a Kleinrock-like cost function ψ_a, (a) plotted on $[0, c_a]$ with linear scale, (b) plotted on $[0, 2c_a]$ with logarithmic scale.

We then use Algorithm 2 and Algorithm 3 to respectively solve the minimum convex cost Steiner flow problem, and the minimum convex cost coded flow problem. Each algorithm runs until either the absolute gap between the current solution and the best known lower-bound is below a threshold $\epsilon = 10^{-2}$ or after 10^4 iterations. The algorithm is coded in Julia 0.3.8 [98, 99]. We use the Julia package "JuMP" [100] to call the open-source mixed-integer programming solver CLP/CBC [101]. When called by Algorithm 2, CBC solves the previosuly presented mixed-integer linear program 2.34 (see also [55]) up to optimality. In both algorithms, the line-search phase is performed thanks to a golden section search algorithm [112] provided by the Julia package "Optim".

4.4.3.2 Undirected networks

Some features of our fifteen instances are summarized in Table 4.2. For each instance, we give the number of vertices $|V|$, the number of edges $|E|$, and the number of receivers $|R|$ in the network. We also provide the demand d, along with the quantity $\beta_\psi(d)$.

Table 4.3 gives, for each undirected instance, information regarding the Steiner flow x returned by Algorithm 2. More precisely, column *UB* indicates the value of the upper-bound obtained by routing the whole demand on a single minimum-cost Steiner tree as

Instance	$\|V\|$	$\|E\|$	$\|R\|$	d	$\beta_\psi(d)$
abilene	12	15	3	5	8020
atlanta	15	22	5	3	6700
france	25	45	6	3	6700
geant	22	36	10	4	7525
germany50	50	88	4	3	6700
giul39	39	86	3	3	6700
india35	35	80	4	3	6700
newyork	16	49	4	5	8020
nobel-eu	28	41	3	5	8020
norway	27	51	5	3	6700
pioro40	40	89	6	2	5050
polska	12	18	5	2	5050
ta1	24	55	6	5	8020
ta2	65	108	4	2	5050
zib54	54	81	4	3	6700

Table 4.2 – Some features of the fifteen undirected instances.

explained in Lemma 6. The next column gives the cost $\psi(x)$ induced by Steiner flow x while column *LB* provides the best known lower-bound ℓ as defined in Equations (4.46)-(4.47). Column *Gap* indicates whether the gap $\psi(x) - \ell$ is smaller than the fixed threshold ϵ. Column *Trees* gives the number of trees actually used by x to convey some positive flow. Column *Congestion* provides the quantity $\max_{e \in E} \frac{f_e}{c_e}$ in percentage, where f is the edge projection of Steiner flow x. Hence a value below 100% implies that x satisfies all capacity requirements. Finally, column *Iterations* gives the number of iterations made by Algorithm 2 while column *Time* provides the algorithm running time expressed in seconds.

Table 4.4 is very similar to Table 4.3 in that it provides, for each undirected instance, information regarding the coded flow x returned by Algorithm 3. Column *LB* gives the lower-bound defined by Equation (4.74) so that column *Gap* is computed accordingly. The quantity $\max_{e \in E} \frac{f_e}{c_e}$ (in percentage), where f is the coding vector of coded flow x is provided in column *Congestion*. A value below 100% implies that x satisfies all capacity requirements.

Observe that, for each undirected instance, the convex cost coding gain g_ψ equals 1.

Instance	UB	$\psi(x)$	LB	Gap	Trees	Congestion (%)	Iterations	Time (s)
abilene	6774.58	20.61	20.6	$\leq \epsilon$	5	86.27	583	18.09
atlanta	6.21	5.74	5.73	$\leq \epsilon$	9	47.2	42	23.83
france	13.63	9.48	9.47	$\leq \epsilon$	51	49.82	635	365.62
geant	3444.13	12.34	12.33	$\leq \epsilon$	18	57.72	86	189.46
germany50	9.74	7.46	7.45	$\leq \epsilon$	17	33.4	65	161.18
giul39	7.71	5.87	5.86	$\leq \epsilon$	17	38.17	300	750.36
india35	7.46	5.23	5.22	$\leq \epsilon$	15	37.5	142	209.59
newyork	11.25	4.36	4.35	$\leq \epsilon$	19	33.93	253	514.37
nobel-eu	11.17	5.45	5.44	$\leq \epsilon$	11	50.36	543	36.29
norway	10.99	6.86	6.85	$\leq \epsilon$	36	33.43	482	2663.28
pioro40	6.8	5.73	5.72	$\leq \epsilon$	17	26.84	120	1274.71
polska	102.55	6.85	6.84	$\leq \epsilon$	11	70.48	72	60.11
ta1	10.5	6.49	6.48	$\leq \epsilon$	32	52.84	128	98.93
ta2	3.3	3.08	3.07	$\leq \epsilon$	7	28.98	19	14.51
zib54	6.16	4.28	4.27	$\leq \epsilon$	8	44.61	19	5.27

Table 4.3 – Features of the Steiner flow x returned by Algorithm 2 for each undirected instance.

Instance	$\psi(x)$	LB	Gap	Congestion (%)	Iterations	Time (s)
abilene	20.61	20.6	$\leq \epsilon$	86.27	583	2.43
atlanta	5.74	5.73	$\leq \epsilon$	47.2	42	0.39
france	9.47	9.46	$\leq \epsilon$	49.81	93	2.0
geant	12.34	12.33	$\leq \epsilon$	57.72	80	2.38
germany50	7.46	7.45	$\leq \epsilon$	33.4	65	2.0
giul39	5.85	5.84	$\leq \epsilon$	39.07	235	5.49
india35	5.23	5.22	$\leq \epsilon$	37.5	142	4.0
newyork	4.36	4.35	$\leq \epsilon$	33.96	252	4.26
nobel-eu	5.45	5.44	$\leq \epsilon$	50.36	543	5.89
norway	6.86	6.85	$\leq \epsilon$	33.41	70	3.3
pioro40	5.73	5.72	$\leq \epsilon$	26.84	120	6.12
polska	6.85	6.84	$\leq \epsilon$	70.48	72	0.6
ta1	6.49	6.48	$\leq \epsilon$	52.84	128	3.65
ta2	3.08	3.07	$\leq \epsilon$	28.98	19	0.72
zib54	4.28	4.27	$\leq \epsilon$	44.61	19	0.51

Table 4.4 – Features of the coded flow x returned by Algorithm 3 for each undirected instance.

4.4.3.3 Bidirected networks

The test bench for bidirected instances is obtained by considering the bidirected network associated to each undirected instance presented in Table 3.1. All other features of the

undirected instances remain unchanged.

Table 4.5 gives, for each bidirected instance, some features regarding the Steiner flow x returned by Algorithm 2.

Instance	UB	$\psi(x)$	LB	Gap	Trees	Congestion (%)	Iterations	Time (s)
abilene	6774.58	20.47	20.46	$\leq \epsilon$	8	86.27	923	10.51
atlanta	6.21	5.69	5.68	$\leq \epsilon$	9	46.31	26	0.42
france	13.63	9.44	9.43	$\leq \epsilon$	46	49.82	241	7.89
geant	3444.13	11.49	11.48	$\leq \epsilon$	58	56.41	556	23.61
germany50	9.74	7.31	7.3	$\leq \epsilon$	26	34.91	374	16.11
giul39	7.71	5.59	5.58	$\leq \epsilon$	17	37.61	342	10.56
india35	7.46	5.11	5.1	$\leq \epsilon$	23	37.34	488	17.68
newyork	11.25	4.22	4.21	$\leq \epsilon$	17	31.43	80	2.13
nobel-eu	11.17	5.31	5.3	$\leq \epsilon$	12	45.97	564	8.81
norway	10.99	6.6	6.59	$\leq \epsilon$	35	30.53	506	16.04
pioro40	6.8	5.68	5.67	$\leq \epsilon$	17	24.02	63	3.96
polska	102.55	6.77	6.76	$\leq \epsilon$	10	70.48	180	2.16
ta1	10.5	6.13	6.12	$\leq \epsilon$	27	52.34	418	14.43
ta2	3.3	3.08	3.07	$\leq \epsilon$	7	28.98	19	0.91
zib54	6.16	4.24	4.23	$\leq \epsilon$	9	44.58	23	0.85

Table 4.5 – Features of the Steiner flow x returned by Algorithm 2 for each bidirected instance.

Similarly, Table 4.6 provides, for each bidirected instance, information on the coded flow x returned by Algorithm 3.

Notice that, for each bidirected instance, the convex cost coding gain g_ψ equals 1.

4.4.3.4 Directed networks

Each directed instance below is actually a bidirected network where, for each link (pair of reverse arcs), the capacity of one of those arcs is multiplied by a coefficient picked in the set $\{1, 2, 3, 5, 10\}$ uniformly at random. This choice preserve the original network connectivity while removing some restrictions found in the previous bidirected setting. All other features of the bidirected instances remain unchanged.

For each directed instance, some features regarding the Steiner flow x returned by Algorithm 2 are provided in Table 4.7.

For each directed instance, some information regarding the coded flow x returned by

Instance	$\psi(x)$	LB	Gap	Congestion (%)	Iterations	Time (s)
abilene	20.47	20.46	$\leq \epsilon$	86.27	925	6.46
atlanta	5.69	5.68	$\leq \epsilon$	46.31	26	0.23
france	9.44	9.43	$\leq \epsilon$	49.82	241	5.81
geant	11.49	11.48	$\leq \epsilon$	56.42	597	18.7
germany50	7.31	7.3	$\leq \epsilon$	34.91	374	13.54
giul39	5.59	5.58	$\leq \epsilon$	37.61	342	9.38
india35	5.11	5.1	$\leq \epsilon$	37.31	488	15.83
newyork	4.22	4.21	$\leq \epsilon$	31.44	78	1.55
nobel-eu	5.31	5.3	$\leq \epsilon$	45.97	564	7.74
norway	6.59	6.58	$\leq \epsilon$	30.02	74	2.01
pioro40	5.68	5.67	$\leq \epsilon$	24.02	63	3.73
polska	6.77	6.76	$\leq \epsilon$	70.48	180	1.6
ta1	6.13	6.12	$\leq \epsilon$	52.3	416	13.01
ta2	3.08	3.07	$\leq \epsilon$	28.98	19	0.85
zib54	4.24	4.23	$\leq \epsilon$	44.58	23	0.7

Table 4.6 – Features of the coded flow x returned by Algorithm 3 for each bidirected instance.

Instance	UB	$\psi(x)$	LB	Gap	Trees	Congestion (%)	Iterations	Time (s)
abilene	6769.3	18.38	18.37	$\leq \epsilon$	5	86.27	2993	22.97
atlanta	5.41	4.46	4.45	$\leq \epsilon$	14	45.89	179	2.49
france	6.63	5.01	5.0	$\leq \epsilon$	7	47.88	82	2.39
geant	8.83	5.9	5.89	$\leq \epsilon$	16	38.68	49	1.86
germany50	5.66	4.69	4.68	$\leq \epsilon$	10	37.5	69	2.8
giul39	5.21	4.28	4.27	$\leq \epsilon$	8	26.09	35	1.04
india35	1.48	1.46	1.45	$\leq \epsilon$	3	23.91	3	0.11
newyork	2.84	2.02	2.01	$\leq \epsilon$	5	31.32	23	0.53
nobel-eu	4.3	3.39	3.38	$\leq \epsilon$	5	40.9	279	4.17
norway	2.64	2.4	2.4	$\leq \epsilon$	6	24.78	9	0.25
pioro40	3.14	2.82	2.81	$\leq \epsilon$	10	27.97	19	7.87
polska	102.08	6.3	6.29	$\leq \epsilon$	14	70.47	1210	14.01
ta1	7.34	4.15	4.14	$\leq \epsilon$	20	46.7	29	0.96
ta2	2.04	1.95	1.94	$\leq \epsilon$	3	29.21	7	0.38
zib54	2.88	2.51	2.5	$\leq \epsilon$	6	29.55	37	1.34

Table 4.7 – Features of the Steiner flow x returned by Algorithm 2 for each directed instance.

Algorithm 3 are given in Table 4.8.

For each directed instance, the convex cost coding gain g_ψ is equal to 1.

Instance	$\psi(x)$	LB	Gap	Congestion (%)	Iterations	Time (s)
abilene	18.38	18.37	$\leq \epsilon$	86.27	2993	15.06
atlanta	4.46	4.45	$\leq \epsilon$	45.89	179	1.87
france	5.01	5.0	$\leq \epsilon$	47.88	82	1.9
geant	5.9	5.89	$\leq \epsilon$	38.68	49	1.52
germany50	4.69	4.68	$\leq \epsilon$	37.5	69	2.41
giul39	4.28	4.27	$\leq \epsilon$	26.09	35	0.91
india35	1.46	1.45	$\leq \epsilon$	23.91	3	0.07
newyork	2.02	2.01	$\leq \epsilon$	31.32	23	0.45
nobel-eu	3.39	3.38	$\leq \epsilon$	40.9	279	3.44
norway	2.4	2.4	$\leq \epsilon$	24.78	9	0.22
pioro40	2.81	2.8	$\leq \epsilon$	26.3	13	0.65
polska	6.3	6.29	$\leq \epsilon$	70.47	1210	10.08
ta1	4.15	4.14	$\leq \epsilon$	46.7	29	0.85
ta2	1.95	1.94	$\leq \epsilon$	29.21	7	0.3
zib54	2.51	2.5	$\leq \epsilon$	29.55	37	1.11

Table 4.8 – Features of the coded flow x returned by Algorithm 3 for each directed instance.

4.4.3.5 Analysis

For each type of information flow, and for each instance, the algorithm stopping criterion is a gap lower than the threshold ϵ. Hence, for each instance and both information flows, the solution returned by the proposed algorithm is always optimal up to ϵ. The benefit of using coding techniques to reduce the overall cost seems to vanish regardless of the network structure. However, it seems possible to find an (almost) optimal coded flow faster than an (almost) optimal Steiner flow. Furthermore, a telecommunication practitioner may have to put a restriction on the number of multicast trees used to convey data, so as to avoid network management issues. Furthermore, although this chapter focuses on the quest for a minimum-cost Steiner or coded flow, a good feasible solution may be sufficient in many practical applications. Such a routing may actually be more convenient to implement due to further restrictions which are not directly taken into account in the proposed models. Consider a network operator who is trying to improve its quality of service by tackling delay issues within a multicast network. Assuming the delay induced by using a channel can be evaluated knowing the load of this channel through a classical Kleinrock-like cost function, it may be more relevant to look for a Steiner flow whose arc projection f satisfies $\max_{a \in A} \frac{f_a}{c_a} < 0.8$ while keeping the number of trees low (say under 5).

The question hence becomes whether it is possible to achieve this configuration by using multicast alone or if some additional coding techniques are required.

4.5 Conclusion

In this chapter we presented two network optimization problems, motivated by practical telecommunication applications, namely the minimum convex cost Steiner flow problem and the minimum convex cost coded flow problem. Together, those two flow models provide a way to evaluate the benefit of using coding techniques in a multicast network for cost (delay) minimization while satisfying a given demand. One can use both problems to define an indicator, the cost coding gain, allowing comparisons between multicast alone versus multicast with coding. We then provide bounds on the *domain* of values one can expect the cost coding gain to take. Those results are summarized in Table 4.9 below. Recall that the greater the value actually taken by the cost coding gain in a given multicast network, the more beneficial network coding techniques would be if implemented in this particular network.

Structure	Linear cost coding gain	Convex cost coding gain
Bidirected	$[1, 2]$	$[1, 2\beta_\psi(d)]$
Undirected	$[1, 2]$	$[1, 2\beta_\psi(d)]$
Directed	$[1, +\infty[$	$[1, +\infty[$

Table 4.9 – Cost coding gain domain for each network type. d is the demand

Finally, it would be interesting to generalize the study undertaken here to the case of multicommodity information flows.

Chapter 5

Minimum cost survivable information flows

5.1 Introduction

5.1.1 Motivation

In the previous chapters, we implicitly assumed that all elements of the given network where perfectly reliable. However, in a real telecommunication network, the failure of an element of the network is the norm rather than the exception. Taking failures into account hence becomes a major issue in the design of any network routing rule. In this chapter we focus on link failures although our models could be easily adapted to cope with node failures or with a mix of the two. For the sake of simplicity, we consider networks with infinite capacity so that it is possible to convey any amount of flow through a channel. We then study the problem where an operator wishes to satisfy a given demand at minimum cost while handling potential failures. This chapter can hence be regarded as an extension of some studies dealing with minimum cost information flows [118, 116], so as to cope with a network prone to failures.

5.1.2 Content

We shall first study the *minimum cost survivable flow problem* which is an extension of the classical minimum cost flow problem so as to deal with *single link failures*. The second problem we shall present, referred to as the *minimum cost survivable Steiner flow problem*,

can be thought of as a generalization of the previous one in the framework of information flows. It is then natural to consider the variant of this Steiner flow problem obtained by allowing coding mechanisms to take place in the telecommunication network, namely the *minimum cost survivable coded flow problem*. We shall then define and study the notion of *survivable cost coding gain* whose role in the present setting will be similar to the one of the cost coding gain as defined in the previous chapter, namely as an indicator of the benefit, in terms of minimum cost, one can expect from using coding techniques in a multicast network.

5.1.3 Minimum cost survivable flow

5.1.3.1 Problem statement

For brevity, we still focus on the case of directed networks which encompasses the case of undirected ones. The instance is made of a network $D = (V, A)$ with a *source* node s, and a *receiver* node r distinct from s. We are given, for each arc a of the network, a non-negative *weight* w_a. Finally, a positive *demand* d is also part of the instance. We assume that there exist at least two arc-disjoint paths from the source to the receiver. We consider an imperfect network which is prone to *single arc failures*. Namely, any arc of the network may fail but we seek protection against at most one such failure at a time (disregarding the case where two or more arcs simultaneously fail). We model the failure of an arc a by removing it from the network, so that it becomes impossible to convey some content through this arc. Recall that a *flow* x is an assignment of a non-negative real value x_p to each simple path p from s to r. Let $\mathcal{P}$ be the set of all such paths, and $\mathcal{P}_a$ be the subset of all paths using arc a. Given a flow x, the failure of arc a actually prevents the flow from delivering some content to the receiver using a path in $\mathcal{P}_a$. The aim is to design a *fault-tolerant* flow which satisfies the demand regardless of whether a failure actually occurs. Our strategy is to look for a *static* flow, which means that no particular action, like re-routing or the activation of backup resources, is undertaken whenever a failure occurs. In this setting, it is required to *over-provision* the flow so as to meet the demand in case of failure. The *nominal throughput* of a flow x is the throughput experienced by the receiver when no failure occurs. Given a flow x, the *residual throughput* $\rho_a(x)$ *of* x *whenever arc*

a fails, is the difference between the *nominal throughput* provided by x and the amount of flow conveyed by x through arc a, namely:

$$\rho_a(x) = \sum_{p \in \mathcal{P}} x_p - \sum_{p \in \mathcal{P}_a} x_p \tag{5.1}$$

or equivalently:

$$\rho_a(x) = \sum_{p \in \mathcal{P} \setminus \mathcal{P}_a} x_p \tag{5.2}$$

Observe that this last quantity is exactly the throughput experienced by the receiver when arc a fails. A flow x is *feasible* if, for each arc a, the residual throughput of x when a fails is at least the demand d. We denote by $\mathcal{X}_\mathcal{P}$ the set of all feasible flows:

$$\mathcal{X}_\mathcal{P} = \left\{ x \in \mathbb{R}_+^\mathcal{P} : \sum_{p \in \mathcal{P} \setminus \mathcal{P}_a} x_p \geq d \; \forall a \in A \right\} \tag{5.3}$$

An operator is interested in reducing the overall bandwidth consumption of the network. It is then natural to look for a flow whose cost, induced by the over-provisioning in bandwidth, is as small as possible. Given a flow x, its *cost* with respect to weight vector w is given by:

$$w(x) = \sum_{a \in A} w_a \sum_{p \in \mathcal{P}_a} x_p \tag{5.4}$$

or equivalently:

$$w(x) = \sum_{p \in \mathcal{P}} \left(\sum_{a \in p} w_a \right) x_p \tag{5.5}$$

This objective function leads to the following problem:

Problem	*Minimum cost (uncapacitated) survivable flow*
Instance	Network $D = (V, A)$, source s, receiver r, weight w_a for each arc a, and positive demand d
Solution	A feasible survivable flow $x \in \mathcal{X}_\mathcal{P}$ in D
Objective	Minimize the cost $w(x)$ of flow x as defined in Equations (5.4)-(5.5)

We denote by χ_F the value of a minimum-cost survivable flow for the considered instance:

$$\chi_F = \min_{x \in \mathcal{X}_\mathcal{P}} w(x) \tag{5.6}$$

Notice that this problem is a natural extension of the minimum (linear) cost (uncapacitated) flow problem so as to cope with single link failures. (The minimum linear cost uncapacitated flow problem amounts to finding a shortest path from the source to the receiver with respect to the given weight vector.)

5.1.3.2 Linear programming formulations

The minimum cost survivable flow problem can be modelled using linear programming:

$$\begin{cases} \chi_F = \min \quad \displaystyle\sum_{p \in \mathcal{P}} \left(\sum_{a \in p} w_a \right) x_p & (5.7) \\ \text{s.t.} \quad \displaystyle\sum_{p \in \mathcal{P} \setminus \mathcal{P}_a} x_p \geq d \quad \forall a \in A & (5.8) \\ \qquad x_p \geq 0 \quad \forall p \in \mathcal{P} & (5.9) \end{cases}$$

where, for each path p, variable x_p stands for the amount of flow routed on p. Observe that, since we may have an exponential number of paths in $\mathcal{P}$, the previous model may also have an exponential number of variables. Constraints (5.8) ensure that flow x satisfies the demand d regardless of whether any single arc fails. We refer to the previous model as the *path formulation* of the minimum cost survivable flow problem, so as to distinguish it from the *arc formulation* of the same problem. The latter leads to the linear program below:

$$\begin{cases} \chi_F = \min \quad \displaystyle\sum_{a \in A} w_a f_a & (5.10) \\ \text{s.t.} \quad \phi - f_a \geq d \quad \forall a \in A & (5.11) \\ \qquad \displaystyle\sum_{a \in \delta^+(v)} f_a - \sum_{a \in \delta^-(v)} f_a = b_v(\phi) \quad \forall v \in V & (5.12) \\ \qquad \phi, f_a \geq 0 \quad \forall a \in A & (5.13) \end{cases}$$

with

$$b_v(\phi) = \begin{cases} \phi & \text{if } v = s \quad (5.14) \\ -\phi & \text{if } v = r \quad (5.15) \\ 0 & \text{otherwise} \quad (5.16) \end{cases}$$

For each arc a, variable f_a stands for the amount of flow going through a, while variable ϕ models the throughput experienced by the receiver r.

5.1.3.3 Complexity and algorithms

The arc formulation of the minimum cost survivable flow problem involves a polynomial number of variables and constraints (with respect to the instance size) so that the problem can be solved in polynomial time. Given an optimal solution f for the arc formulation,

an optimal solution x for the path formulation can be built in polynomial time thanks to the technique presented in the proof of Lemma 1: Compute a path decomposition of the simple flow induced by vector f thanks to the algorithm presented by Ahuja et al. [7, Theorem 3.5] which runs in polynomial time. We refer the interested reader to the survey by Orlowski and Pióro [119, 120] for an alternative approach using column generation.

5.2 Minimum cost survivable Steiner flow

5.2.1 Problem statement

We shall now study the extension of the previous problem to a multicast network. For brevity, we focus our study on the case of a directed network since it encompasses the case of an undirected one. However, we will make a distinction between a directed network (which may be bidirected) prone to *single arc failures* and a bidirected network prone to *single link failures*. In the latter case we assume that *both arcs* of the link are simultaneously failing while in the former case only one arc fails. The instance is made of a network $D = (V, A)$ with a *source* s, a set R of *receivers*, a non-negative *weight* w_a for each arc a, and a positive *demand* d. We assume that there exist at least two arc-disjoint paths from the source to each receiver. Recall that a *Steiner flow* x is an assignment of a non-negative real value x_t to each Steiner tree t spanning s and R. Let $\mathcal{T}$ denote the set of all such trees, and, for each arc a, let $\mathcal{T}_a$ be the subset of all trees using arc a. Given a Steiner tree t and a receiver r, we denote by p_t^r the unique path from s to r in t. For each arc a and each receiver r, let $\mathcal{T}_a^r$ be the set of all Steiner trees t using a to convey flow toward r, namely $a \in p_t^r$. We assume again that any arc a of the network may fail. We still model such a failure by removing the corresponding arc from the network. However the impact of a failure on a Steiner flow differs from the one of the same failure on a classical flow. Indeed, the failure of arc a prevents any Steiner tree t from using the failed arc to convey flow from the source s to any receiver r connected to s through a. In other words, when arc a fails, any receiver r such that $a \in p_t^r$ loses its access to the content conveyed by Steiner tree t. However, a receiver lying in the same connected component (which is a sub-tree of t) as the source after the removal of the failed arc will still receive the flow conveyed by t. Figure 5.1 provides an example. This time, the aim is

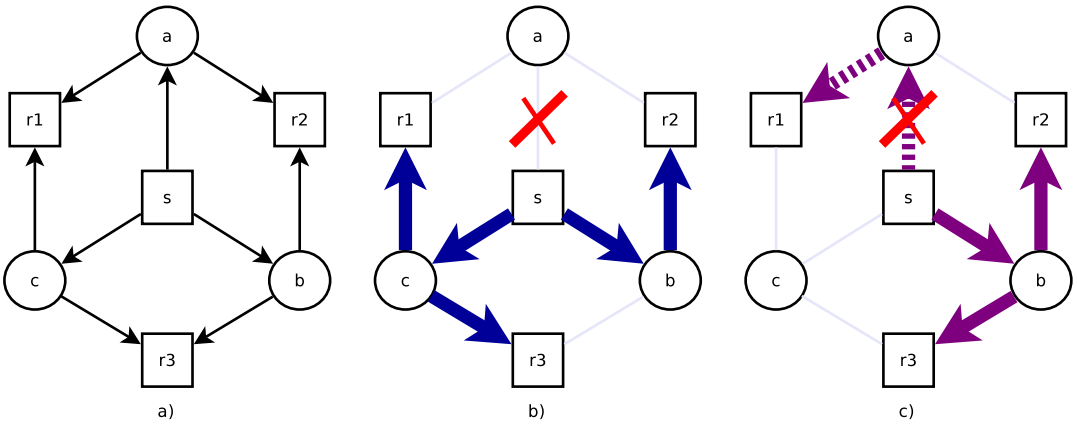


Figure 5.1 – a) A directed network with one source and three receivers. b) A Steiner tree which does not use the arc (s, a) is left unaffected by the failure. c) A Steiner tree using the arc (s, a) is unable to convey flow to receiver r_1, as emphasized by the dotted line, but it can still deliver flow to receivers r_2 and r_3.

to design a *fault-tolerant* Steiner flow which satisfies the demand regardless of whether a failure actually occurs. Our strategy remains to look for a *static* Steiner flow, requiring no particular action or counter-measure (like re-routing or the activation of backup resources) to handle a failure. It is again required to *over-provision* the Steiner flow so as to meet the demand in case of failure. We call *nominal throughput* of a Steiner flow x the throughput experienced by each receiver when no failure occurs. Notice that, in our model, a failure may cause a huge difference in the throughputs experienced among all receivers. Given a Steiner flow x, the *residual throughput* $\rho_a^r(x)$ *of* x *with respect to receiver* r*, whenever arc* a *fails*, is the difference between the *nominal throughput* provided by x and the amount of flow conveyed by x, through arc a, toward receiver r, namely:

$$\rho_a^r(x) = \sum_{t \in \mathcal{T}} x_t - \sum_{t \in \mathcal{T}_a^r} x_t \tag{5.17}$$

or equivalently:

$$\rho_a^r(x) = \sum_{t \in \mathcal{T} \setminus \mathcal{T}_a^r} x_t \tag{5.18}$$

Observe that $\rho_a^r(x)$ is exactly the throughput experienced by receiver r when arc a fails. A Steiner flow x is *feasible* if, for each arc a and each receiver r, the residual throughput of x experienced by r when a fails is at least the demand d. We denote by $\mathcal{X}_\mathcal{T}$ the set of

all feasible Steiner flows:

$$\mathcal{X}_{\mathcal{T}} = \left\{ x \in \mathbb{R}_+^{\mathcal{T}} : \sum_{t \in \mathcal{T} \setminus \mathcal{T}_a^r} x_t \geq d \, \forall a \in A, r \in R \right\} \tag{5.19}$$

As previously explained, an operator should look for a Steiner flow whose cost, induced by the over-provisioning in bandwidth, is as small as possible. Given a Steiner flow x, its *cost* with respect to weight vector w is given by:

$$w(x) = \sum_{a \in A} w_a \sum_{t \in \mathcal{T}_a} x_t \tag{5.20}$$

or equivalently:

$$w(x) = \sum_{t \in \mathcal{T}} \left(\sum_{a \in t} w_a \right) x_t \tag{5.21}$$

This objective function naturally leads to the problem below:

Problem	*Minimum cost (uncapacitated) survivable Steiner flow*
Instance	Network $D = (V, A)$, source s, set of receivers R, weight w_a for each arc a, and positive demand d
Solution	A feasible survivable Steiner flow $x \in \mathcal{X}_{\mathcal{T}}$ in D
Objective	Minimize the cost $w(x)$ of Steiner flow x as defined in Equations (5.20)-(5.21)

We denote by χ_S the value of a minimum-cost survivable Steiner flow for the considered instance:

$$\chi_S = \min_{x \in \mathcal{X}_{\mathcal{T}}} w(x) \tag{5.22}$$

Observe that this problem is a natural extension of the minimum (linear) cost (uncapacitated) Steiner flow problem in order to cope with single link failures. (The minimum linear cost uncapacitated Steiner flow problem amounts to finding a minimum-cost Steiner tree from the source to the receiver with respect to the given weight vector.)

5.2.2 A detailed example

We consider again the directed butterfly network which is depicted on Figure 5.2 a). We are interested in solving the minimum cost survivable Steiner flow problem under unit demand, namely $d = 1$, where the weight of each arc a of the network is set to $w_a = 1$. A minimum-cost survivable Steiner flow $\bowtie$ is depicted on Figure 5.2 b). Steiner flow $\bowtie$ uses

two trees, each one routing the whole demand d so that its overall cost is 10. Thanks to Steiner flow $\varkappa$, each receiver experiences a throughput of value d regardless of whether any arc is removed from the network.

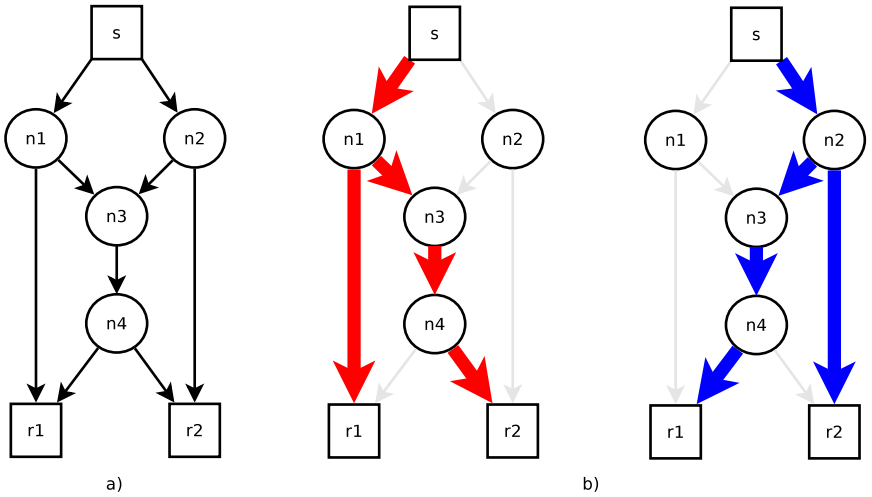


Figure 5.2 – a) The directed butterfly network. b) A minimum-cost survivable Steiner flow using two trees in the directed butterfly network. Each tree conveys the whole demand d so that the cost of the induced Steiner flow is 10.

5.2.3 Linear programming formulation

The minimum cost survivable Steiner flow problem can be modelled using linear programming:

$$\begin{cases} \chi_S = \min \quad \displaystyle\sum_{t \in \mathcal{T}} \left(\sum_{a \in t} w_a \right) x_t & (5.23) \\ \text{s.t.} \quad \displaystyle\sum_{t \in \mathcal{T} \setminus \mathcal{T}_a^r} x_t \geq d \quad \forall a \in A, r \in R & (5.24) \\ \qquad x_t \geq 0 \quad \forall t \in \mathcal{T} & (5.25) \end{cases}$$

where, for each Steiner tree t, variable x_t stands for the amount of flow routed on t. Notice that the previous model may, as the previous linear program, have an exponential number of variables. Constraints (5.24) ensure that Steiner flow x satisfies the demand d regardless of whether any single arc fails. If one is interested by the model of *single link failures* in a bidirected network $B = (V, L)$, Constraints (5.24) should be replaced by the following

ones:

$$\sum_{t\in\mathcal{T}} x_t - \left(\sum_{t\in\mathcal{T}^r_{a_1}} x_t + \sum_{t\in\mathcal{T}^r_{a_2}} x_t\right) \geq d \quad \forall \ell = \{a_1, a_2\} \in L, r \in R \tag{5.26}$$

where we rest on the fact that a Steiner tree is using at most *one* arc of each link, so that the two sets $\mathcal{T}^r_{a_1}$ and $\mathcal{T}^r_{a_2}$ are disjoint for any link $\ell = \{a_1, a_2\}$ and any receiver r. Let z be the vector of dual variables associated to Constraints (5.24). The dual of the present linear program is:

$$\begin{cases} \chi_S = \max \quad d \sum_{a\in A}\sum_{r\in R} z^r_a & (5.27) \\ \text{s.t.} \quad \sum_{r\in R}\sum_{a\in A\setminus p^r_t} z^r_a \leq \sum_{a\in t} w_a \quad \forall t \in \mathcal{T} & (5.28) \\ \qquad z^r_a \geq 0 \quad \forall a \in A, r \in R & (5.29) \end{cases}$$

Observe that finding a minimum-cost survivable flow amounts to solving a fractional covering problem. Hence the dual of the minimum cost survivable Steiner flow problem is a fractional packing problem. It will be convenient to rewrite Constraints (5.28) as follows:

$$\sum_{a\in A}\sum_{r\in R} z^r_a \leq \sum_{a\in t} w_a + \sum_{r\in R}\sum_{a\in p^r_t} z^r_a \tag{5.30}$$

5.2.4 Complexity and algorithms

We shall now study the complexity of the minimum cost survivable Steiner flow problem. Unfortunately, we are not able to either exhibit an algorithm running in polynomial time, or to provide a reduction to a known NP-hard problem. We will now consider the *separation problem* associated to the dual of the present problem. This separation problem, also referred to as the *pricing problem* in the following, can be stated as follows:

Problem	*Pricing problem*
Instance	Network $D = (V, A)$, source s, set of receivers R, non-negative weight w_a for each arc a, and non-negative weight z^r_a for each arc a and each receiver r
Solution	A Steiner tree t spanning the source and all receivers in D
Objective	Minimize the cost of the tree which is given by $\sum_{a\in t} w_a + \sum_{r\in R}\sum_{a\in p^r_t} z^r_a$

Observe that if vector z equals 0, solving the pricing problem amounts to finding a minimum-cost Steiner tree with respect to weights vector w. However, we are also in-

terested by the other case where at least one component of vector z is positive. We shall actually focus on the special case where the weights vector z satisfies the equality:

$$\sum_{a \in A} \sum_{r \in R} z_a^r = 1 \tag{5.31}$$

The study of this restriction of the pricing problem is motivated by the role it will play in the next chapter. We shall now prove that the pricing problem where the instance satisfies Equation (5.31) is NP-hard in both directed and undirected networks.

5.2.4.1 Directed networks

We begin with the easier case of a directed network.

Theorem 24. *The pricing problem is NP-hard in directed networks.*

Proof. We perform a reduction from the two node-disjoint paths problem.

Problem	*Two node-disjoint paths*
Instance	Directed network $D = (V, A)$, four distinct vertices i_1, j_1, i_2, and j_2
Solution	Two node-disjoint paths, one from i_1 to j_1, the other one from i_2 to j_2 in D
Objective	Decide whether two such paths exist

This problem is NP-complete [121]. The reduction goes as follows: given an instance of the two node-disjoint paths problem, we build an instance of the pricing problem by adding a source s and two receivers r_1 and r_2, along with arcs (s, i_1), (s, i_2), (j_1, r_1), and (j_2, r_2). We set $w_a = 0$ for each arc a of the resulting digraph except (j_1, r_1) and (j_2, r_2), with $w_{(j_1,r_1)} = \frac{1}{2}$ and $w_{(j_2,r_2)} = \frac{1}{2}$. We also set $z_a^r = 0$ for each arc a of the resulting graph and each of the two receivers, except for $z_{(s,i_1)}^{r_2} = \frac{1}{2}$ and $z_{(s,i_2)}^{r_1} = \frac{1}{2}$. Notice that, the sum over all arcs and the two receivers of the weights z_a^r equals 1, as required. A scheme of this construction is depicted on Figure 5.3. We shall prove now that there exist two node-disjoint paths, one from i_1 to j_1, the other one from i_2 to j_2, if and only if there is a Steiner arborescence rooted at the source and spanning both receivers, with weight 1. To see why, observe that, if there exist two node-disjoint paths, one from i_1 to j_1 and another from i_2 to j_2, then we can use those two paths to build a Steiner arborescence rooted at s and spanning $\{r_1, r_2\}$ with weight exactly 1. Conversely, suppose such a Steiner

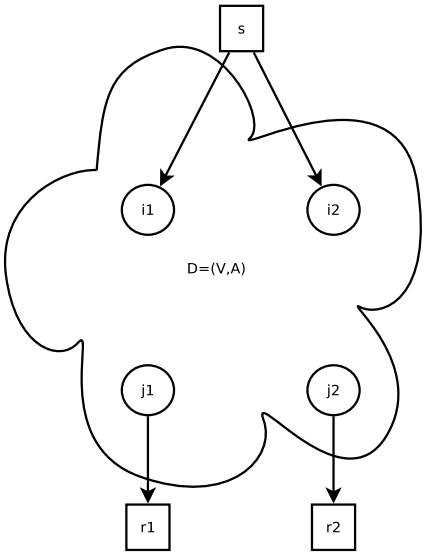


Figure 5.3 – Scheme of the instance construction used in the proof of Theorem 24.

arborescence t with weight 1 exists in our construction. Notice that such an arborescence uses arcs (j_1, r_1) and (j_2, r_2). Furthermore, the path $p_t^{r_1}$ necessarily uses the arc (s, i_1), hence the node i_1, otherwise it would have to pay the cost $z^{r_1}_{(s,i_2)}$ which would lead to an overall weight strictly greater than 1. Similarly, the path $p_t^{r_2}$ goes through node i_2. Thus tree t induces two paths, one from i_1 to j_1 and another one from i_2 to j_2. Moreover, those paths are node-disjoint since t is an arborescence. □

5.2.4.2 Undirected networks

We now turn our attention to the case of an undirected network.

Theorem 25. *The pricing problem is NP-hard in undirected networks.*

Proof. We perform a reduction from SAT with two or three literals per clause, and such that each variable occurs at most three times in the whole formula (which is assumed to be in conjunctive normal form).

Problem	*Restricted SAT*
Instance	Set of variables, set of clauses with two or three literals per clause such that each variable occurs at most three times
Solution	An assignment of the value TRUE or FALSE to each variable such that each clause evaluates at TRUE
Objective	Decide whether such an assignment exists

This problem is NP-complete [122]. Observe that we can safely assume each variable appears at most twice and its negation also appears at most twice in any instance (otherwise it is possible to eliminate this variable by setting it to either TRUE or FALSE). The reduction is as follows: to each variable we associate a gadget, depicted on Figure 5.4 a), and to each clause we associate another gadget, represented in Figure 5.4 b). We then define an undirected network by chaining all variable gadgets, chaining all clause gadgets, and adding a source s and two receivers r_1 and r_2, as depicted in Figure 5.5. Consider the gadget associated to a variable x. The bold edges on the left side of the gadget represent the first and second occurrences respectively of this variable as a positive literal (with respect to an arbitrary but fixed ordering) of the clauses, while the bold edges on the right stand for the first and second occurrences respectively of its negation (with respect to the same ordering). Finally, assume variable x appears in clause c and this occurrence is the i^{th} in the whole Boolean formula (again with respect to the same ordering). Then we add two edges between the gadget of clause c and the i^{th} bold edge in the left side of the gadget of variable x, such that each vertex of the clause gadget is incident to exactly one of these two edges. If, instead, the negation of variable x appears in clause c and this occurrence is the i^{th} in the whole Boolean formula, then we add two edges between the gadget of clause c and the i^{th} bold edge in the right side of the gadget of variable x. We do this for all variables and all clauses. Also add one edge e_1 between the source and the first variable gadget, another edge e_2 between the source and the first clause gadget, one edge e_3 between the last variable gadget and receiver r_1, and one last edge e_4 between the last clause gadget and receiver r_2, as depicted in Figure 5.5. Let n be the number of variables, and m the number of clauses. Set $w_e = 0$ for each edge e of the network except e_3 and e_4 with $w_{e_3} = \frac{1}{2}$ and $w_{e_4} = \frac{1}{2}$. Set the weight $z_{e_1}^{r_2} = \frac{1}{4}$, and the weight $z_{e_2}^{r_1} = \frac{1}{4}$. For each edge between two consecutive clause gadgets, set its weight with respect to receiver r_1 to $\frac{1}{4(m-1)}$ (we assume $m \geq 2$). For each non-bold edge of a variable gadget, set its weight with respect to receiver

r_2 to $\frac{1}{24n}$. All other weights of the network are set to 0. Since there are six non-bold edges for each variable gadget, we have the sum over all edges and over both receivers of the weights z which equals 1, as required. We shall prove now that the formula is satisfiable if and only if there is a Steiner tree spanning both receivers and the source, with weight 1. First assume the formula is satisfiable. This implies there exists a path p_2 between s and r_2 using edge e_2, all edges between two consecutive clause gadgets, some bold edges associated to an assignment of either TRUE or FALSE to a variable or its negation, and for each clause c, two edges between the gadget associated to c and the gadget of one variable x which appears in c. Thus, this path is zigzagging between clause gadgets and variable gadgets. Observe that, for each gadget of an assigned variable, all bold edges used by this path lies on the same side of the gadget, either left or right, since the formula is satisfiable. Thus, it is possible to build a path p_1 between s and r_1 using edge e_1, all edges between two consecutive variable gadgets and all edges of either the left side or the right side of each variable gadget, such that the two paths are node-disjoint. Combining those two paths gives a Steiner tree spanning the source and both receivers. Furthermore this tree has a weight of value 1 since no edge with a positive weight, with respect to the corresponding receiver, is used in the tree except for edges e_3 and e_4. Conversely, assume there exists a Steiner tree t, spanning the source and both receivers, with weight 1 in the network. This Steiner tree induces two node-disjoint paths $p_t^{r_1}$ and $p_t^{r_2}$ between the source and receivers r_1 and r_2 respectively. The path $p_t^{r_1}$ must cross all variable gadgets, and actually it must stay on the same side, either left or right, of each variable gadget, since otherwise the weight of the tree would be strictly greater than 1. By the same argument, the path $p_t^{r_2}$ uses all vertices corresponding to clause gadgets and it only uses bold edges in any crossed variable gadget. Furthermore, the path $p_t^{r_2}$ uses bold edges of the same side, either left or right, of each variable gadget it intersects with, since $p_t^{r_1}$ and $p_t^{r_2}$ are node-disjoint. This implies one can deduce a valid assignment of Boolean to some variables, so that each clause is evaluated at TRUE, by following the path $p_t^{r_1}$. Hence, the formula is satisfiable, which concludes the proof. □

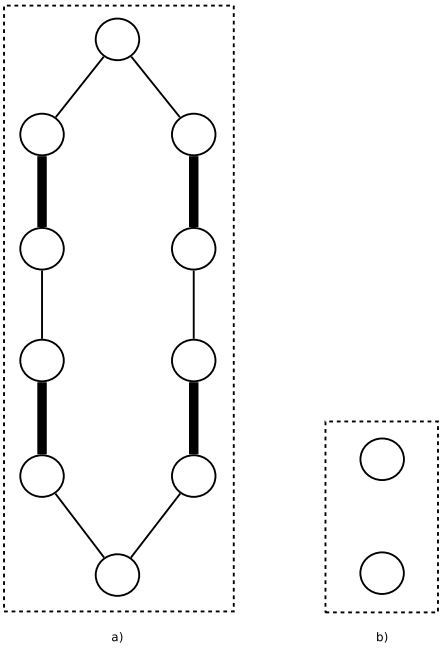


Figure 5.4 – Gadget a) for a variable, and b) for a clause, used in the proof of Theorem 25.

5.2.4.3 Wrap up

As we have just seen, the pricing problem is NP-hard both in directed and undirected networks. However, observe that all coefficients in the objective function of the dual of the present problem share the same value d, see Equation (5.27). It is thus impossible to apply the celebrated theorem of Grötschel, Lovász, and Schrijver on the equivalence between polynomial-time separation and polynomial-time optimization for convex polytopes [90]. The previous discussion only emphasizes that, unless $P = NP$, it is impossible to devise an algorithm running in polynomial time by combining the ellipsoid method with an oracle solving the pricing problem. Beware that, it may still be possible to find an algorithm which would solve the minimum cost survivable Steiner flow problem in polynomial time despite the two last theorems (for instance, by using another linear programming formulation). We refer the interested reader to the paper by Nace et al. [123] for a non-trivial example of a problem whose extended linear programming formulation admits an NP-hard separation problem while the problem itself can be solved in polynomial time. Hence, we shall leave the complexity of finding a minimum-cost survivable Steiner flow as an open question:

Conjecture 1. *We believe that the minimum cost survivable Steiner flow problem is NP-*

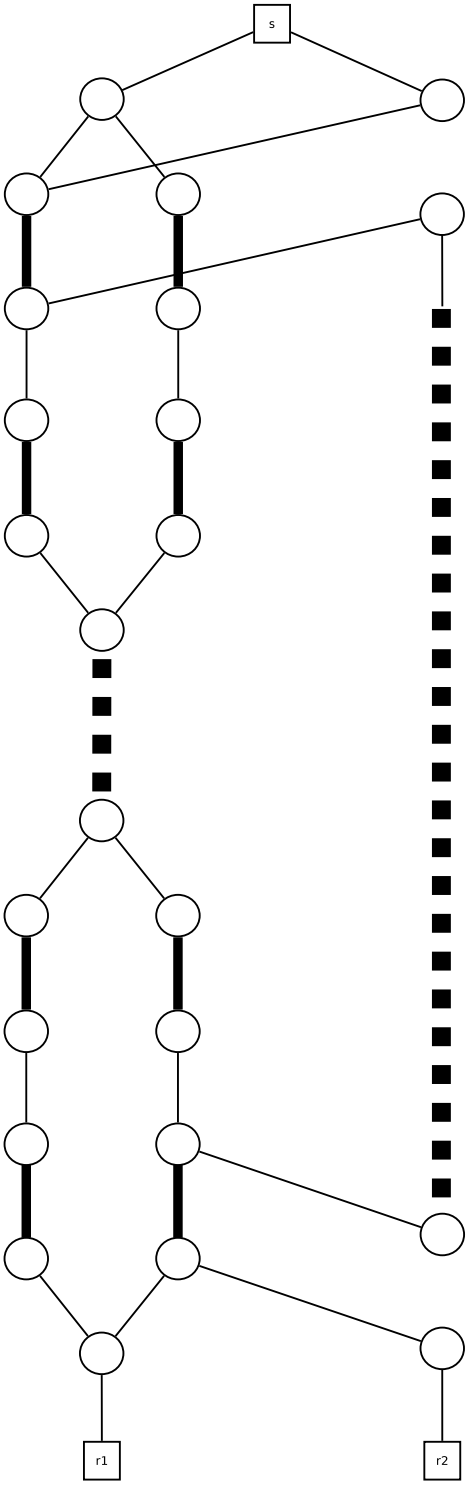


Figure 5.5 – Scheme of the instance construction used in the proof of Theorem 25.

hard.

5.2.5 Redundant Steiner trees

We shall now study an interesting restriction of the minimum cost survivable Steiner flow problem. Notice that a survivable Steiner flow requires *at least* two distinct trees so as to ensure that regardless of whether a failure actually occurs, each receiver remains reachable from the source. Consider the setting where a survivable Steiner flow is further restricted to use *at most* two distinct Steiner trees. This setting is motivated by the complexity of managing a bunch of trees. This last restriction implies that, for each receiver, the two paths from the source to this receiver, induced by the couple of routing trees (one path per tree), have to be arc-disjoint. Formally, we are looking for an element in the set $\mathcal{S}$ defined by:

$$\mathcal{S} = \left\{(t_1, t_2) \in \mathcal{T}^2 : p^r_{t_1} \cap p^r_{t_2} = \emptyset \; \forall r \in R\right\} \tag{5.32}$$

Given a couple $(t_1, t_2) \in \mathcal{S}$, one obtains a survivable Steiner flow by routing the whole demand d on each tree. Furthermore, the overall cost of this Steiner flow is simply the demand d times the sum of the costs of the two trees. Thus the restricted problem amounts to finding a couple of Steiner trees in set $\mathcal{S}$ whose total cost with respect to weight vector w is as small as possible. This last problem can be summarized as follows:

Problem	*Redundant Steiner trees*
Instance	Network $D = (V, A)$, source s, set of receivers R, weight w_a for each arc a
Solution	A couple of Steiner trees $(t_1, t_2) \in \mathcal{S}$ in D
Objective	Minimize the overall cost $\sum_{a \in t_1} w_a + \sum_{a \in t_2} w_a$

This problem, along with some extensions to simultaneously handle single node failures, have been extensively studied in the multicast literature. Although the present problem is intractable in the worst case, various heuristic algorithms have been devised, returning practically low-cost solutions. An approximation algorithm with worst-case ratio $2|R|$ is presented by Bejerano et al. in [124]. We refer the interested reader to [125, 126] and references therein for a more in-depth discussion regarding the present problem. It is also worth mentioning that the problem of finding a couple of minimum-cost redundant

spanning trees in a directed network can be solved to optimality in polynomial time [127]. The integer linear program below can be used to find a minimum-cost pair of redundant Steiner trees, rooted at s and spanning all vertices in R, in a directed network $D = (V, A)$ with weight w_a on each arc a (a valid model for undirected networks can be inferred from this one).

$$
\begin{cases}
\min \quad \sum_{a \in A} w_a \mathbb{x}_a + \sum_{a \in A} w_a \mathbb{x}_a & (5.33) \\
\text{s.t.} \quad \sum_{a \in \delta^+(v)} \mathbb{f}_a^r - \sum_{a \in \delta^-(v)} \mathbb{f}_a^r = b_v^r(1) \quad \forall r \in R, v \in V & (5.34) \\
\quad \sum_{a \in \delta^+(v)} f_a^r - \sum_{a \in \delta^-(v)} f_a^r = b_v^r(1) \quad \forall r \in R, v \in V & (5.35) \\
\quad \mathbb{f}_a^r \leq \mathbb{x}_a \quad \forall a \in A, r \in R & (5.36) \\
\quad f_a^r \leq \mathbb{x}_a \quad \forall a \in A, r \in R & (5.37) \\
\quad \mathbb{f}_a^r + f_a^r \leq 1 \quad \forall a \in A, r \in R & (5.38) \\
\quad \mathbb{f}_a^r \in \{0,1\}, f_a^r \in \{0,1\}, \mathbb{x}_\ell \in \{0,1\}, \mathbb{x}_\ell \in \{0,1\} \quad \forall a \in A, r \in R & (5.39)
\end{cases}
$$

with

$$
b_v^r(\phi) = \begin{cases} \phi & \text{if } v = s \quad (5.40) \\ -\phi & \text{if } v = r \quad (5.41) \\ 0 & \text{otherwise} \quad (5.42) \end{cases}
$$

For each arc a, and each receiver r, variable $\mathbb{f}_a^r$, respectively f_a^r, is set to 1 if and only if arc a is used to connect source s to receiver r in the first, respectively second, tree. For each arc a, variable $\mathbb{x}_a$, respectively $\mathbb{x}_a$, is set to 1 if and only if a is part of the first, respectively second, tree. Constraints (5.34)-(5.35) ensure that both trees are connected and spanning the source along with all receivers. Constraints (5.36)-(5.37) prevent an arc a from being used as part of a tree unless the arc is actually picked as part of this tree. Finally, Constraints (5.38) are the coupling constraints between the pair of trees, enforcing that said couple of trees lies in set $\mathcal{S}$ as defined in Equation (5.32).

5.2.6 Computing a minimum cost survivable Steiner flow by column generation

One can solve the minimum cost survivable Steiner flow problem by mean of column generation techniques. In order to get an initial survivable Steiner flow, one can compute a pair of redundant Steiner trees by using the integer linear program presented in the previous section, then route the whole demand d on each of those two trees. Given dual vectors z, the *reduced cost* $\mathbb{r}_t$ associated to variable x_t (which we shall refer to as the reduced cost of tree t by a slight abuse of language) has the following expression:

$$\mathbb{r}_t = \sum_{a \in A} \sum_{r \in R} z_a^r - \left(\sum_{a \in t} w_a + \sum_{r \in R} \sum_{a \in p_t^r} z_a^r \right) \tag{5.43}$$

As previously explained, the *pricing problem* amounts to looking for a Steiner tree whose cost, with respect to both weights vectors w and z, is minimum. The following mixed-integer linear program can be used to solve this pricing problem to optimality. The model below is provided for a directed network $D = (V, A)$ with weight w_a on each arc a (a valid model for undirected networks can be inferred from this one).

$$\begin{cases} \min \sum_{a \in A} w_a x_a + \sum_{r \in R} \sum_{a \in A} z_a^r f_a^r & (5.44) \\ \text{s.t.} \sum_{a \in \delta^+(v)} f_a^r - \sum_{a \in \delta^-(v)} f_a^r = b_v^r(1) \quad \forall r \in R, v \in V & (5.45) \\ f_a^r \leq x_a \quad \forall a \in A, r \in R & (5.46) \\ \sum_{a \in \delta^-(v)} x_a \leq 1 \quad \forall v \in V & (5.47) \\ x_a \in \{0,1\} \quad \forall a \in A & (5.48) \\ f_a^r \geq 0 \quad \forall a \in A, r \in R & (5.49) \end{cases}$$

with

$$b_v^r(\phi) = \begin{cases} \phi & \text{if } v = s \quad (5.50) \\ -\phi & \text{if } v = r \quad (5.51) \\ 0 & \text{otherwise} \quad (5.52) \end{cases}$$

For each arc a, binary variable x_a takes value 1 if a is part of the tree, and value 0 otherwise. Flow variable f_a^r will take value 1 if the unique directed path from s to r in the tree is using arc a, and value 0 otherwise. Constraints (5.45) ensure that each receiver

is connected to the source in the tree, while Constraints (5.46) force each arc used by the tree to convey some flow to be actually part of the tree. Finally, Constraints (5.47) ensure that the connected subgraph induced by all arcs actually conveying some positive flow is indeed a tree. Observe the similarity between this mixed-integer linear program and one of the classical formulation for the minimum cost Steiner tree problem presented in [55]. If, by solving this problem, we find a tree with a positive reduced cost, this tree represents a new column to be added to the master problem, which is then solved again. If we are unable to exhibit such a tree, then it means that we have obtained an optimal solution.

We shall now study the problem of finding a coded flow with minimum cost.

5.3 Minimum cost survivable coded flow

5.3.1 Problem statement

Departing from the previous chapters, we will make a distinction between directed networks on the one hand, and bidirected or undirected networks on the other hand. To summarize the distinction, the behavior of an *uncapacitated* bidirected network prone to *single link failures* is identical to the one of an undirected network prone to *single edge failures*, assuming that the weights of both arcs of any link are equal. We first focus on the case of directed networks. The setting is exactly the same as in the previous problem: The instance is made of a network $D = (V, A)$ with a *source* s, a set R of *receivers*, a non-negative *weight* w_a for each arc a, and a positive *demand* d. We assume again that there exist at least two arc-disjoint paths from the source to each receiver. Recall that a *coded flow* x is an assignment of a non-negative real value x_p to each simple path p from the source to a receiver. Let $\mathcal{P}$ be the set of all such paths. For each receiver r, let $\mathcal{P}^r$ denote the set of all simple paths from s to r. Notice that $\mathcal{P} = \bigcup_{r \in R} \mathcal{P}^r$. Finally, for each arc a and each receiver r, let $\mathcal{P}^r_a$ be the subset of $\mathcal{P}^r$ made of all paths using a. We still assume that any arc a of the network may fail, an event which is again modelled by removing the failed arc from the network. The impact of a failure on a coded flow is quite close from the one of the same failure on a classical flow. The failure of arc a prevents any path p from using the failed arc to convey flow from the source to the corresponding receiver. In other

words, given a coded flow x, the failure of arc a actually prevents the flow from delivering some content to receiver r using a path in $\mathcal{P}_a^r$. Similarly to the previous cases, the aim is to design a *fault-tolerant* coded flow which satisfies the demand regardless of whether a failure actually occurs. Our strategy is still to look for a *static* coded flow, requiring no particular action or counter-measure (like re-routing or the use of backup resources) to handle a failure. We postpone the design of a *fault-tolerant code* to a devoted section, later in this chapter. It is yet required to *over-provision* the coded flow so as to meet the demand in case of failure. Given a coded flow x, the *nominal throughput of a receiver* r is the throughput experienced by r when no failure occurs. Observe that, here also, a failure may cause a huge difference in the throughputs experienced among all receivers. Given a coded flow x, the *residual throughput* $\rho_a^r(x)$ *of* x *with respect to receiver* r*, whenever arc* a *fails*, is the difference between the *nominal throughput* of receiver r (as provided by x) and the fraction of this nominal throughput conveyed through arc a, namely:

$$\rho_a^r(x) = \sum_{p \in \mathcal{P}^r} x_p - \sum_{p \in \mathcal{P}_a^r} x_p \tag{5.53}$$

or equivalently:

$$\rho_a^r(x) = \sum_{p \in \mathcal{P}^r \setminus \mathcal{P}_a^r} x_p \tag{5.54}$$

Notice that $\rho_a^r(x)$ is exactly the throughput experienced by receiver r when arc a fails. A coded flow x is *feasible* if, for each arc a and each receiver r, the residual throughput of x experienced by r when a fails is at least the demand d. We denote by $\mathcal{X}_\mathcal{P}$ the set of all feasible coded flows:

$$\mathcal{X}_\mathcal{P} = \left\{ x \in \mathbb{R}_+^\mathcal{P} : \sum_{p \in \mathcal{P}^r \setminus \mathcal{P}_a^r} x_p \geq d \; \forall a \in A, r \in R \right\} \tag{5.55}$$

As already mentioned, an operator should look for a coded flow whose cost, induced by the over-provisioning in bandwidth, is as small as possible. Recall that, for a coded flow x and an arc a, the bandwidth consumed by x on a is given by:

$$h_a = \max_{r \in R} \sum_{p \in \mathcal{P}_a^r} x_p \tag{5.56}$$

where vector h is called the *coding vector* of coded flow x. Given a coded flow x along with its coding vector h, the *cost* of x with respect to weight vector w is given by:

$$w(x) = \sum_{a \in A} w_a h_a \tag{5.57}$$

This objective function leads to the following problem:

Problem	*Minimum cost (uncapacitated) survivable coded flow*
Instance	Network $D = (V, A)$, source s, set of receivers R, weight w_a for each arc a, and positive demand d
Solution	A feasible survivable coded flow $x \in \mathcal{X}_{\mathcal{P}}$ in D
Objective	Minimize the cost $w(x)$ of coded flow x as defined in Equation (5.57)

We denote by χ_C the value of a minimum-cost survivable coded flow for the considered instance:

$$\chi_C = \min_{x \in \mathcal{X}_{\mathcal{P}}} w(x) \tag{5.58}$$

Observe that this problem is a natural extension of the minimum (linear) cost (uncapacitated) coded flow problem so as to handle single link failures.

5.3.2 Example continued

Consider the instance made of the directed butterfly network depicted on Figure 5.2 a) with demand $d = 1$, and a weight $w_a = 1$ for each arc a. A minimum-cost survivable coded flow χ for this instance is depicted on Figure 5.6. The coded flow χ uses two paths routing the whole demand d to convey the data from the source to each receiver, so that its overall cost is 9. Thanks to coded flow χ, each receiver experiences a throughput of value d regardless of whether any arc is removed from the network.

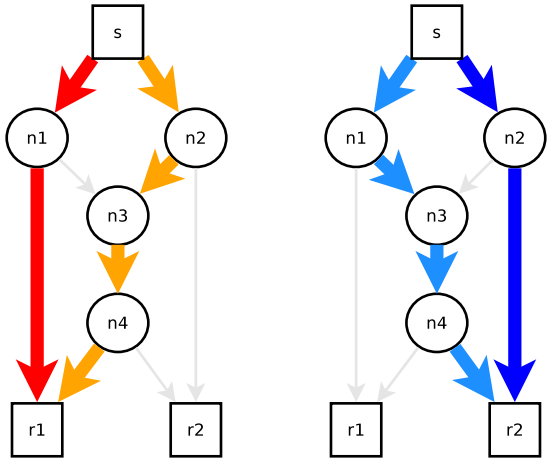


Figure 5.6 – A minimum-cost survivable coded flow χ using exactly two paths to route the data from the source to each receiver. Each path is routing the whole demand d so that the overall cost induced by coded flow χ is 9.

5.3.3 Linear programming formulations

5.3.3.1 Directed networks

In a directed network $D = (V, A)$, the minimum cost survivable coded flow problem can be formalized using linear programming:

$$\begin{cases} \chi_C = \min \sum_{a \in A} w_a h_a & (5.59) \\ \text{s.t.} \quad \sum_{p \in \mathcal{P}^r \setminus \mathcal{P}^r_a} x_p \geq d \quad \forall a \in A, r \in R & (5.60) \\ \quad h_a \geq \sum_{p \in \mathcal{P}^r_a} x_p \quad \forall a \in A, r \in R & (5.61) \\ \quad h_a, x_p \geq 0 \quad \forall a \in A, p \in \mathcal{P} & (5.62) \end{cases}$$

For each path p, variable x_p stands for the amount of flow routed on p, while, for each arc a, variable h_a models the amount of coded flow going through a. Observe that we may again end up with an exponential number of variables to deal with. Constraints (5.60) ensure that coded flow x satisfies the demand d regardless of whether any single arc fails. From this *path formulation* of the minimum cost survivable coded flow problem one can easily build an alternative *arc formulation* of the same problem. The linear program below is naturally associated to this arc formulation of the problem:

$$\begin{cases} \chi_C = \min \quad \sum_{a \in A} w_a h_a & (5.63) \\ \text{s.t.} \quad \phi^r - f_a^r \geq d \quad \forall a \in A, r \in R & (5.64) \\ \quad \sum_{a \in \delta^+(v)} f_a^r - \sum_{a \in \delta^-(v)} f_a^r = b_v^r(\phi^r) \quad \forall r \in R, v \in V & (5.65) \\ \quad h_a \geq f_a^r \quad \forall a \in A, r \in R & (5.66) \\ \quad \phi^r, f_a^r, h_a \geq 0 \quad \forall a \in A, r \in R & (5.67) \end{cases}$$

with

$$b_v^r(\phi) = \begin{cases} \phi & \text{if } v = s \quad (5.68) \\ -\phi & \text{if } v = r \quad (5.69) \\ 0 & \text{otherwise} \quad (5.70) \end{cases}$$

For each arc a and each receiver r, variable f_a^r stands for the amount of flow going from source s to receiver r through a, and variable h_a accounts for the amount of coded flow on arc a, whereas variable ϕ^r models the throughput experienced by receiver r.

5.3.3.2 Bidirected and undirected networks

We will now study the minimum cost survivable coded flow problem in undirected or bidirected networks. For the sake of simplicity, we focus on the case of a bidirected network $B = (V, L)$ prone to *single link failures* (without loss of generality as already explained in the beginning of the present section). As in the case of a directed network, the problem of

interest can be formalized using linear programming:

$$\begin{cases} \chi_C = \min \quad \displaystyle\sum_{\ell \in L} w_\ell (h_{a_1} + h_{a_2}) & (5.71) \\ \text{s.t.} \quad \displaystyle\sum_{p \in \mathcal{P}^r} x_p - \left(\sum_{p \in \mathcal{P}^r_{a_1}} x_p + \sum_{p \in \mathcal{P}^r_{a_2}} x_p \right) \geq d \quad \forall \ell = \{a_1, a_2\} \in L, r \in R & (5.72) \\ \quad h_a \geq \displaystyle\sum_{p \in \mathcal{P}^r_a} x_p \quad \forall a \in \ell, \ell \in L, r \in R & (5.73) \\ \quad h_a, x_p \geq 0 \quad \forall a \in \ell, \ell \in L, p \in \mathcal{P} & (5.74) \end{cases}$$

where we rest on the fact that a simple path is using at most *one* arc of each link, so that the two sets $\mathcal{P}^r_{a_1}$ and $\mathcal{P}^r_{a_2}$ are disjoint for any link $\ell = \{a_1, a_2\}$ and any receiver r. As in the previous case, for each path p, variable x_p stands for the amount of flow routed on p, while, for each arc a, variable h_a models the amount of coded flow going through a. Constraints (5.72) ensure that coded flow x satisfies the demand d regardless of whether any single *link* fails. Since this *path formulation* of the minimum cost survivable coded flow problem involves an exponential number of variables, it appears quite natural to introduce the *arc formulation* of the same problem to which is associated the following linear program:

$$\begin{cases} \chi_C = \min \quad \displaystyle\sum_{\ell \in L} w_\ell (h_{a_1} + h_{a_2}) & (5.75) \\ \text{s.t.} \quad \phi^r - (f^r_{a_1} + f^r_{a_2}) \geq d \quad \forall \ell = \{a_1, a_2\} \in L, r \in R & (5.76) \\ \quad \displaystyle\sum_{a \in \delta^+(v)} f^r_a - \sum_{a \in \delta^-(v)} f^r_a = b^r_v(\phi^r) \quad \forall r \in R, v \in V & (5.77) \\ \quad h_a \geq f^r_a \quad \forall a \in \ell, \ell \in L, r \in R & (5.78) \\ \quad \phi^r, f^r_a, h_a \geq 0 \quad \forall a \in \ell, \ell \in L, r \in R & (5.79) \end{cases}$$

where again

$$b^r_v(\phi) = \begin{cases} \phi & \text{if } v = s \quad (5.80) \\ -\phi & \text{if } v = r \quad (5.81) \\ 0 & \text{otherwise} \quad (5.82) \end{cases}$$

As in the case of a directed network, for each arc a and each receiver r, variable f^r_a stands for the amount of flow going from source s to receiver r through a, variable h_a accounts for the amount of coded flow on arc a, while variable ϕ^r models the throughput experienced by receiver r.

5.3.4 Complexity and algorithms

Regardless of the network structure (directed or bidirected / undirected), the linear program obtained by considering the arc formulation of the minimum cost survivable coded flow problem involves a polynomial number of (continuous) variables and constraints (with respect to the instance size). Hence, it is possible to compute an arc decomposition of a minimum-cost survivable coded flow in polynomial time. Given such an arc decomposition f, a path decomposition x can be built in polynomial time thanks to the technique presented in the proof of Lemma 1: For each receiver r, let f^r be the vector of components f^r_a over each arc a of the network. Observe that vector f^r is the arc projection of a *simple* flow between the source and receiver r. For each receiver r, compute a path decomposition of this simple flow using the algorithm presented by Ahuja et al. [7, Theorem 3.5] which runs in polynomial time. The concatenation of all those path decompositions (one per receiver) is clearly a valid path decomposition of the coded flow.

5.3.5 Coding scheme

Although one can compute the value of a minimum cost survivable coded flow, as previously defined, it is unclear whether this value has any meaning. Indeed, achieving the target demand thanks to a coded flow requires not only to take routing decisions but also involves the design of a coding scheme. This issue has already been addressed by Koetter and Médard [59, Theorems 11], for the more general case where multiple arcs may simultaneously fail, from which we deduce the following corollary:

Corollary 10. *Given a* directed acyclic network $D = (V, A)$ *with source s, set of receivers R, and target throughput d, there exists a static linear coding scheme, common to all single arc failure patterns, and ensuring a throughput of value d.*

Hence, not only can one meet the target demand, but this can be done using only one coding scheme for handling all failure patterns. This last property is particularly appealing, since it means no update of the way coding operations are performed in the network is required in order to cope with a failure. An extension of [59, Theorem 11] to the case of a network with delays is mentioned at the end of the same paper. An important feature

of a coding scheme is the size of the finite field where coding operations are performed. Building a code whose underlying finite field has a size as small as possible is a desirable property since it ensures that the coding process remains tractable. The price of a static coding scheme common to all failure patterns, in terms of an upper-bound on the size of the required finite field, is emphasized by the following corollary, which is a direct application of [59, Theorem 12]:

Corollary 11. *Given a* directed acyclic network $D = (V, A)$ *with source s, set of receivers R, and target throughput d, there exists a static linear coding scheme, common to all single arc failure patterns, where all operations are performed in a finite field $\mathbb{F}_{2^q}$ with $q \leq \lceil \log_2(1 + |A||R|d) \rceil$.*

Those results above only give a proof of existence of a coding scheme. The following corollary from [65, Theorem 11] gives a more practical result:

Corollary 12. *Given a* directed acyclic network $D = (V, A)$ *with source s, set of receivers R, and target throughput d, a static linear coding scheme, common to all single arc failure patterns, where all operations are performed in a finite field $\mathbb{F}_{2^q}$ with $q \leq \lceil \log_2(1+|A||R|d) \rceil$, can be found in expected time $O(|A|^2(|R|d^2 + \max_{v \in V} |\delta^-(v)|))$.*

The upper bound on the size of the required finite field is further improved in [128]. Regarding networks with cycles, it is still possible to design a fault-tolerant coding scheme by using more complicated coding techniques, as explained in [59].

5.4 Features of the information flows

5.4.1 Survivable cost coding gain

We shall now study the impact of implementing coding techniques in the present framework. We are led to ask the following question:

Question 4. *Is there an incentive, with respect to the overall survivable cost, to use network coding in a multicast network?*

Similarly to the previously studied frameworks, we introduce the notion below so as to facilitate comparisons:

Definition 5. *Given a network $D = (V, A)$ with source s, set of receivers R, non-negative weight w_a for each arc a, along with positive demand d, the* survivable cost coding gain g_χ*, provided by network coding over multicast alone, is the ratio of the cost of a minimum-cost survivable Steiner flow over the one of a minimum-cost survivable coded flow:*

$$g_\chi = \frac{\chi_S}{\chi_C} \tag{5.83}$$

Recall that we only consider networks where there is at least two arc-disjoint paths from the source to any receiver. Once combined with the absence of explicit capacity requirements, it should be clear that this last property implies that both minimum-cost information flow problems are feasible. We shall further assume that the quantity χ_C is positive so that the previous definition of the *survivable cost coding gain* g_χ is well-grounded. The next theorem shows that adding the possibility to perform coding operations in a multicast network never impedes the survivable cost one has to pay to use this network:

Theorem 26. *Given a network $D = (V, A)$ with source s, set of receivers R, non-negative weight w_a for each arc a, along with positive demand d, the survivable cost coding gain g_χ is at least* 1*.*

The proof of Theorem 26 is very similar to the one of Theorem 3 and can be found in Appendix C.1.4. Notice that, with respect to the survivable cost, there is an incentive to use network coding only if $g_\chi > 1$. We shall now study the survivable cost coding gain of some networks.

5.4.2 Features of the survivable cost coding gain

5.4.2.1 Single receiver

As always, we first focus on the setting where there is only one receiver. In this case both the minimum cost survivable Steiner flow problem and the minimum cost survivable coded flow problem become equivalent to the minimum cost survivable flow problem. This immediately implies the following lemma:

Lemma 14. *Given a network $D = (V, A)$ with source s, a single receiver r distinct from s, non-negative weight w_a for each arc a, along with positive demand d, the survivable cost coding gain g_χ equals* 1*.*

5.4.2.2 Bidirected and undirected networks

The next theorem provides an upper-bound on the survivable cost coding gain of an undirected network.

Theorem 27. *Given an undirected network $G = (V, E)$ with source s, set of receivers R, non-negative weight w_e for each edge e, along with positive demand d, the value of a minimum-cost survivable Steiner flow is at most $2|E|$ times the one of a minimum-cost survivable coded flow, namely $g_\chi \leq 2|E|$.*

Proof. For each edge e, we denote by $G - e$ the undirected network obtained from G by deleting e while keeping the remaining of the instance unchanged. By a slight abuse of notation, we still denote by w the weight vector where the component associated to a given edge e is removed. For each edge e, let ψ_S^e, ψ_C^e, and $g_\psi^e = \frac{\psi_S^e}{\psi_C^e}$ be respectively the cost of a minimum-cost Steiner flow, the cost of a minimum-cost coded flow, and the cost coding gain associated to the instance defined by considering network $G - e$ with source s, set of receivers R, truncated weight vector w, along with demand d. Observe that for any edge e, the three quantities ψ_S^e, ψ_C^e, and g_ψ^e are well-defined from our assumption that there is at least two edge-disjoint paths from the source to each receiver in network G, so that there is at least one path between the source and any receiver r in network $G - e$. For each edge e, let t^e be a minimum-cost Steiner tree in network $G - e$ with respect to the (truncated) weight vector w. Beware that for each edge e, Steiner tree t^e does *not* contain e. Recall from our previous discussion regarding the minimum linear cost Steiner flow problem, that a minimum-cost Steiner flow can always be designed by routing the whole demand on a minimum-cost Steiner tree with respect to the same weight vector. This implies that for each edge $\mathbb{e}$, we have $\psi_S^{\mathbb{e}} = d \sum_{e \in t^{\mathbb{e}}} w_e$. Let T be the collection of the Steiner trees t^e over all edges. We claim that it is always possible to build a survivable Steiner flow x by routing the whole demand d on each tree in T. To see why, observe that the failure of an edge e would leave the Steiner tree t^e unaffected, so that each receiver would still get a throughput of value d thanks to tree t^e. The cost of the induced Steiner flow x is:

$$w(x) = d \sum_{t \in T} \sum_{e \in t} w_e = \sum_{e \in E} \psi_S^e \tag{5.84}$$

This implies that:

$$\chi_S \leq w(x) \tag{5.85}$$

$$\leq |E| \max_{e \in E} \psi_S^e \tag{5.86}$$

$$\leq 2|E| \max_{e \in E} \psi_C^e \tag{5.87}$$

where the last inequality comes from the upper bound of 2 on the linear cost coding gain g_ψ of any undirected network [42, mentioned before Theorem 6]. Finally, observe that the inequality $\max_{e \in E} \psi_C^e \leq \chi_C$ is valid since, for any edge e, a minimum-cost coded flow in the network $G - e$ only protects against the failure of edge e while a minimum-cost survivable coded flow should simultaneously protect against the failure of any edge. Hence, $\chi_S \leq 2|E|\chi_C$ or equivalently $g_\chi \leq 2|E|$. □

As previously explained, looking for a minimum-cost survivable information flow in an uncapacitated bidirected network is equivalent to seeking for the same minimum-cost survivable information flow in an uncapacitated undirected network $G = (V, E)$. As a direct consequence, the upper bound of $2|E|$ also applies for bidirected networks.

5.4.2.3 Directed networks

The following theorem highlights that the survivable cost coding gain can be arbitrarily large in a directed network. It can be regarded as an extension to the present survivable framework of a result due to Maheshwar et al. [85, regarding Conjecture 8.1] regarding the cost coding gain in directed networks.

Theorem 28. *For any real number $\eta \geq 1$, there exists an infinite family of directed networks with weight either 1 or 0 on each arc, such that the ratio between the value of a minimum-cost survivable Steiner flow and the one of a minimum-cost survivable coded flow satisfies $g_\chi \geq \eta$.*

Proof. We will exhibit a family of directed networks with unbounded survivable cost coding gain. We again use the family of directed combination networks introduced in [83, 84] to show that the coding gain can be arbitrarily large. Recall that given two positive integers n and k with $k \in [n]$, the directed combination network $C(n, k)$ is made of three layers of

vertices: the first layer with the source s, while the second layer is made of n intermediate nodes identified with $[n]$, whereas the third layer consists of a set R of $\binom{n}{k}$ receivers, indexed by the subsets of $[n]$ of size k. Those three layers are linked together by one arc from the source to each intermediate node, and one arc from an intermediate node to each receiver whose index (a subset of $[n]$) contains the index of this intermediate node (an element of $[n]$). The weight of each arc a from the first layer to the second one is set to $w_a = 1$, while each arc a from the second layer to the third one has a weight $w_a = 0$. Finally, we shall assume that the demand is $d = 1$, without loss of generality. The directed combination network $C(n, k)$, for $n = 4$ and $k = 3$, is depicted on Figure (5.7) (a). We first give an upper-bound on the cost χ_C of a minimum-cost coded flow. Observe that there are exactly k arc-disjoint paths from the source to each receiver. Hence, it is possible to design a survivable coded flow by routing $1/(k-1)$ units of flow along each of the k paths from s to r, for each receiver r. The cost of this survivable coded flow with respect to weights vector w is $n/(k-1)$. This implies that $\chi_C \leq n/(k-1)$. Although it is not required here, it can be shown by linear programming duality that this last inequality is actually an equality. We shall now provide a lower-bound on the cost χ_S of a minimum-cost survivable Steiner flow. Formally, we build a feasible solution $\mathbb{z}$ to the dual of the linear program associated to the minimum cost survivable Steiner flow problem. For each arc a, and each receiver r, we set:

$$\mathbb{z}_a^r = \begin{cases} \dfrac{n-k+1}{(k-1)\binom{n}{k}} & \text{if } a \in \delta^-(r) \quad (5.88) \\ 0 & \text{otherwise} \quad (5.89) \end{cases}$$

First notice that:

$$\sum_{a \in A} \sum_{r \in R} \mathbb{z}_a^r = \sum_{r \in R} \sum_{a \in \delta^-(r)} \mathbb{z}_a^r \quad (5.90)$$

$$= \sum_{r \in R} \frac{k(n-k+1)}{(k-1)\binom{n}{k}} \quad (5.91)$$

$$= \frac{k(n-k+1)}{(k-1)} \quad (5.92)$$

where the second equality is obtained by observing that, for each receiver r, there is exactly k arcs in the set $\delta^-(r)$, while the third equality is valid since there is $\binom{n}{k}$ receivers in the

directed combination network $C(n,k)$. Now, observe that for each Steiner tree t:

$$\sum_{r\in R}\sum_{a\in p_t^r} \mathbb{z}_a^r = \sum_{r\in R}\sum_{a\in p_t^r\cap\delta^-(r)} \mathbb{z}_a^r \tag{5.93}$$

$$= \sum_{r\in R} \frac{n-k+1}{(k-1)\binom{n}{k}} \tag{5.94}$$

$$= \frac{n-k+1}{(k-1)} \tag{5.95}$$

The second equality is valid because, for each receiver r, there is only one arc in the set $p_t^r \cap \delta^-(r)$. The third equality comes again from the number $\binom{n}{k}$ of receivers in the directed combination network $C(n,k)$. Recall from Lemma 3 that in such a network, a Steiner tree t requires at least $n-k+1$ intermediate nodes in order to span all receivers. Hence, for each Steiner tree t:

$$\sum_{a\in t} w_a \geq n-k+1 \tag{5.96}$$

Combining this last inequality with Equation (5.92) and Equation (5.95) we get for each Steiner tree t:

$$\sum_{a\in t} w_a + \sum_{r\in R}\sum_{a\in p_t^r} \mathbb{z}_a^r \geq n-k+1+\frac{n-k+1}{(k-1)} \tag{5.97}$$

$$= \frac{k(n-k+1)}{(k-1)} \tag{5.98}$$

$$= \sum_{a\in A}\sum_{r\in R} \mathbb{z}_a^r \tag{5.99}$$

so that vector $\mathbb{z}$ satisfies Equation (5.30) and is thus a feasible solution to the dual linear program with value $\frac{k(n-k+1)}{(k-1)}$. By weak duality in linear programming, we get $\chi_S \geq k(n-k+1)/(k-1)$. Although it is not needed here, notice that it is possible to define a survivable Steiner flow by suitably picking k Steiner trees in $C(n,k)$, each tree using exactly $n-k+1$ intermediate nodes in the network, then setting a flow of value $1/(k-1)$ on each tree. The cost induced by this survivable Steiner flow is then $k(n-k+1)/(k-1)$ so that the last inequality is actually an equality. We refer the interested reader to Figure (5.7) (b) for an example. Combining the latest lower-bound on χ_S with the previous upper-bound on χ_C leads to:

$$g_\chi \geq \frac{k(n-k+1)}{n} \tag{5.100}$$

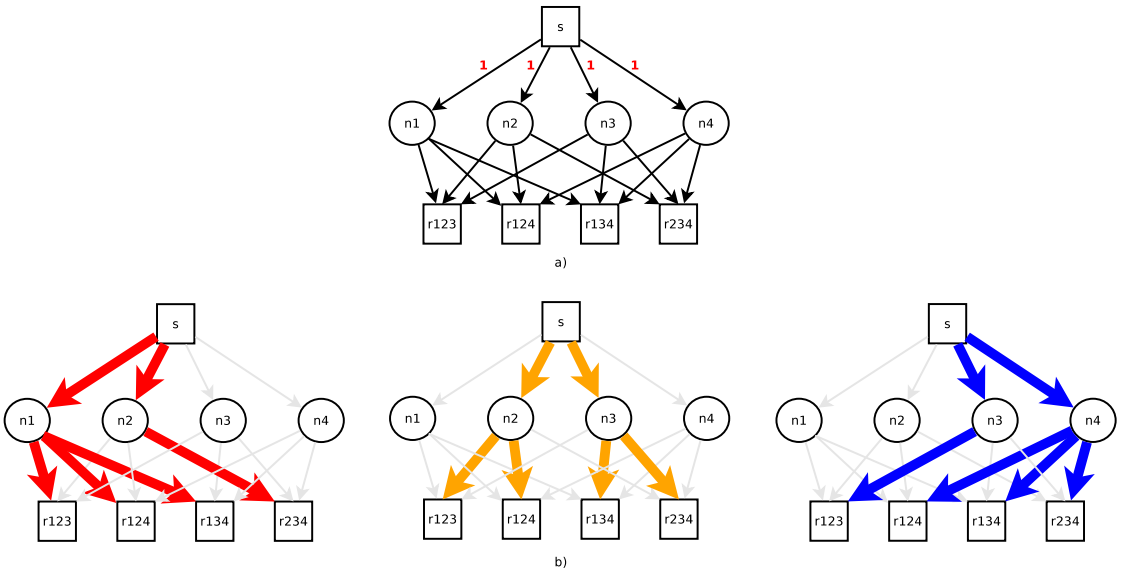


Figure 5.7 – a) The directed combination network $C(n,k)$ where $n=4$ and $k=3$. When positive, the weight of each arc a is indicated in red near a. b) A Steiner flow x made of three supporting trees, each one routing $1/2$ units of flow, in the directed combination network $C(4,3)$. The cost induced by Steiner flow x is 3.

Setting $n=2k$ yields $g_\chi \geq (k+1)/2$. This last quantity is unbounded when the parameter k grows to infinity. Finally, given any real number $\eta \geq 1$, the network $C(2k,k)$ with $k=\lceil 2\eta-1 \rceil$ satisfies all requirements of the theorem. □

5.4.3 Experimental evaluation of the survivable cost coding gain

5.4.3.1 Setting

We use the same fifteen network topologies taken from the SNDlib library [97] as in the previous chapters. We also keep the source and the receivers as in the previous chapter. The weight of each channel is picked in the set [10] (of all integers between 1 and 10) uniformly at random. The demand is set to $d=1$ without loss of generality.

We compute each optimal Steiner flow by coupling a column generation algorithm with a subroutine solving the pricing problem thanks to the mixed-integer linear program 5.49 previously presented. Each optimal coded flow problem is computed by solving its associated arc formulation thanks to a linear programming solver. All algorithms are implemented in Julia 0.3.8 [98, 99]. We use the Julia package *JuMP* [100] to call the open-source

(mixed-integer) linear programming solver *CLP/CBC* [101].

5.4.3.2 Undirected networks

For each instance, the number of vertices $|V|$, the number of edges $|E|$, and the number of receivers $|R|$ of the associated network is provided in Table 5.1

Instance	$\|V\|$	$\|E\|$	$\|R\|$
abilene	12	15	3
atlanta	15	22	5
france	25	45	6
geant	22	36	10
germany50	50	88	4
giul39	39	86	3
india35	35	80	4
newyork	16	49	4
nobel-eu	28	41	3
norway	27	51	5
pioro40	40	89	6
polska	12	18	5
ta1	24	55	6
ta2	65	108	4
zib54	54	81	4

Table 5.1 – Some features of the fifteen undirected instances.

Table 5.2 gives, for each undirected instance, information regarding the minimum-cost survivable Steiner flow returned by the column generation algorithm. More precisely, column χ_S^0 indicates the cost of the best pair of redundant Steiner trees, whereas column χ_S provides the cost of the optimal Steiner flow. Column *Trees* gives the number of trees actually used by the optimal Steiner flow to convey some positive flow. Finally, column *Iterations* gives the number of iterations made by the column generation algorithm while column *Time* provides the associated running time expressed in seconds.

Table 5.3 is similar to Table 5.2 in that it provides, for each undirected instance, information regarding the minimum-cost survivable coded flow returned by the linear programming solver. Column χ_C gives the value of the optimal coded flow while column *Time* provides the associated running time expressed in seconds.

Notice that, for each undirected instance, the survivable cost coding gain g_χ equals 1.

Instance	χ_S^0	χ_S	Trees	Iterations	Time (s)
abilene	63.0	63.0	2	6	0.88
atlanta	79.0	79.0	2	9	4.62
france	89.0	89.0	2	21	8.21
geant	70.0	70.0	2	13	79.19
germany50	67.0	55.0	3	52	292.38
giul39	44.0	38.5	3	22	7.23
india35	43.0	40.0	6	21	8.57
newyork	26.0	23.0	3	14	10.07
nobel-eu	40.0	40.0	2	6	1.27
norway	73.0	69.5	5	45	63.87
pioro40	120.0	101.33	9	96	1782.7
polska	51.5	53.0	2	10	2.27
ta1	44.0	44.0	2	9	4.28
ta2	40.0	40.0	2	20	15.0
zib54	65.0	54.5	5	12	5.88

Table 5.2 – Features of the minimum-cost survivable Steiner flow returned by the column generation algorithm for each undirected network.

Instance	χ_C	Time (s)
abilene	63.0	0.01
atlanta	79.0	0.01
france	89.0	0.02
geant	68.0	0.03
germany50	55.0	0.03
giul39	38.5	0.02
india35	40.0	0.02
newyork	23.0	0.01
nobel-eu	40.0	0.01
norway	69.5	0.02
pioro40	101.33	0.04
polska	51.5	0.01
ta1	44.0	0.02
ta2	40.0	0.03
zib54	54.5	0.02

Table 5.3 – Features of the minimum-cost survivable coded flow returned by the linear programming solver for each undirected network.

Recall that the survivable cost coding gain of a bidirected network prone to single *link* failures is the same as the one of the associated undirected network with respect to single edge failures.

5.4.3.3 Directed networks

Each directed instance below is actually a bidirected network, built from the associated undirected one. In the directed setting, we consider single arc failures instead of single edge or link failures. All other features of the undirected instances remain unchanged.

Table 5.4 gives, for each undirected instance, information regarding the minimum-cost survivable Steiner flow returned by the column generation algorithm. More precisely, column χ_S^0 indicates the cost of the best pair of redundant Steiner trees, whereas column χ_S provides the cost of the optimal Steiner flow. Column *Trees* gives the number of trees actually used by the optimal Steiner flow to convey some positive flow. Finally, column *Iterations* gives the number of iterations made by the column generation algorithm while column *Time* provides the associated running time expressed in seconds.

Instance	χ_S^0	χ_S	Trees	Iterations	Time (s)
abilene	63.0	63.0	2	6	0.7
atlanta	81.0	81.0	2	12	0.33
france	89.0	89.0	2	16	0.79
geant	68.0	70.0	2	16	1.29
germany50	67.0	55.0	3	50	3.69
giul39	44.0	38.5	3	24	1.23
india35	43.0	40.0	3	22	1.88
newyork	26.0	23.0	3	14	0.53
nobel-eu	40.0	40.0	2	6	0.14
norway	73.0	69.5	7	39	2.85
pioro40	120.0	101.33	6	100	47.3
polska	51.5	51.5	2	13	0.24
ta1	44.5	44.5	2	7	0.39
ta2	40.0	40.0	2	14	1.13
zib54	65.0	54.5	3	13	0.78

Table 5.4 – Features of the minimum-cost survivable Steiner flow returned by the column generation algorithm for each directed network.

Table 5.5 gives, for each directed instance, information regarding the minimum-cost survivable coded flow returned by the linear programming solver. Column χ_C gives the value of the optimal coded flow while column *Time* provides the associated running time expressed in seconds.

Observe that, for each directed instance, the survivable cost coding gain g_χ equals 1.

Instance	χ_C	Time (s)
abilene	63.0	0.04
atlanta	81.0	0.01
france	89.0	0.02
geant	68.0	0.03
germany50	55.0	0.03
giul39	38.5	0.01
india35	40.0	0.02
newyork	23.0	0.01
nobel-eu	40.0	0.01
norway	69.5	0.02
pioro40	101.33	0.06
polska	51.5	0.01
ta1	44.5	0.02
ta2	40.0	0.03
zib54	54.33	0.02

Table 5.5 – Features of the minimum-cost survivable coded flow returned by the linear programming solver for each directed network.

5.4.3.4 Analysis

For each type of information flow, and for each instance, the proposed algorithm finds an optimal solution in a matter of seconds. It seems there is few incentives to use coding techniques, with respect to the survivable cost coding gain, regardless of the network structure. Since the considered networks are sparse, it seems often possible to design a minimum-cost survivable Steiner flow supported by a pair of redundant Steiner trees.

5.5 Conclusion

In this chapter we considered two network optimization problems, the minimum cost survivable Steiner flow problem and the minimum cost survivable coded flow problem. Together, those two models provide a way to evaluate the benefit, in terms of cost reduction, of using coding techniques in a multicast network prone to single link failure. One can use both problems to define an indicator, called the *survivable cost coding gain*, allowing comparisons between multicast alone versus multicast with coding. We then provide bounds on the *domain* of values one can expect the cost coding gain to take. Those results

are summarized in Table 5.6 below. Recall that the greater the value actually taken by the survivable cost coding gain in a given multicast network, the more beneficial network coding techniques would be if implemented in this particular network.

Structure	Cost coding gain	Survivable cost coding gain
Bidirected	$[1, 2]$	$[1, 2\|L\|]$
Undirected	$[1, 2]$	$[1, 2\|E\|]$
Directed	$[1, +\infty[$	$[1, +\infty[$

Table 5.6 – Survivable cost coding gain domain for each network type.

An operator is often forced to look for a trade off between various conflicting objectives like ensuring low congestion in the network while over-provisioning the amount of conveyed flow so as to diminish the impact of a failure on the throughputs actually experienced by the receivers. In this regard, it would be interesting to combine the models developed in this chapter along with those presented in the previous one, since solving a minimum convex cost survivable information flow problem could be done by combining the Frank-Wolfe method with a subroutine looking for a minimum-cost survivable information flow.

Chapter 6

Maximum survivable information flows

6.1 Introduction

6.1.1 Motivation

In this chapter we pursue the study undertaken in the previous chapter regarding survivable information flows. At the same time, the present work can be considered as an extension of the one done in the second chapter. Indeed, we shall now study the problem of maximizing the worst case residual throughput experienced by a set of receivers in a network prone to single link failures. The present chapter can be regarded as an attempt to evaluate the *fragility* of information flows, with respect to single link failures.

6.1.2 Content

We will first study the *maximum survivable flow problem* which is an extension of the classical maximum flow problem so as to handle *single link failures*. The second problem we shall present, referred to as the *maximum survivable Steiner flow problem*, can be thought of as a generalization of the maximum survivable flow problem to the setting of information flows. We then consider the variant of this Steiner flow problem which appears when coding mechanisms are allowed in the network, namely the *maximum survivable coded flow problem*. We will finally define and study the notion of *survivable coding gain* which is similar to the coding gain previously presented. The survivable coding gain is an indicator

which evaluates the impact, in terms of maximum residual throughput, one can expect from deploying coding techniques in a multicast network.

6.1.3 Maximum survivable flow

6.1.3.1 Problem statement

As before, we focus on the case of directed networks which encompasses the case of undirected ones. The instance is made of a network $D = (V, A)$ with a *source* node s, a *receiver* node r distinct from s and a capacity c_a for each arc a. Recall that a *flow* x is an assignment of a non-negative real value x_p to each simple path p from s to r, and that x is *feasible* if it satisfies the capacity constraints (the flow conservation at each vertex is implied by the paths decomposition). We denote by $\mathcal{X}_\mathcal{P}$ the set of all feasible flows:

$$\mathcal{X}_\mathcal{P} = \left\{ x \in \mathbb{R}_+^\mathcal{P} : \sum_{p \in \mathcal{P}_a} x_p \leq c_a \, \forall a \in A \right\} \tag{6.1}$$

where $\mathcal{P}$ is the set of all simple paths between the source and the receiver, and $\mathcal{P}_a$ is the subset of all paths in P using arc a.

Now we assume that any arc a of the network may fail. We model this event by removing the corresponding arc from the network. The impact of the failure of arc a on a given feasible flow x is to prevent the content routed along any path p using a from being conveyed to the receiver. Given a feasible flow x, we define, for each arc a, the *residual throughput* $\rho_a(x)$ *of* x *whenever arc* a *fails*, as the quantity:

$$\rho_a(x) = \sum_{p \in \mathcal{P}} x_p - \sum_{p \in \mathcal{P}_a} x_p \tag{6.2}$$

or equivalently:

$$\rho_a(x) = \sum_{p \in \mathcal{P} \setminus \mathcal{P}_a} x_p \tag{6.3}$$

Observe that the first term in the right-hand side of Equation (6.2) is the *nominal throughput* of flow x, the amount of flow which can be conveyed from the source to the receiver when no failure occurs, while the second term (in the right-hand side of the same equation) is exactly the amount of flow on arc a. Hence $\rho_a(x)$ is exactly the amount of flow experienced by the receiver r when arc a fails. Given a feasible flow x, the *residual throughput* $\rho(x)$

of x is defined as the minimum, over all possible arc failures, of the residual throughput $\rho_a(x)$ associated to the failure of arc a:

$$\rho(x) = \min_{a \in A} \rho_a(x) \tag{6.4}$$

We are now ready to define formally our problem:

Problem	*Maximum survivable flow*
Instance	Network $D = (V, A)$, source s, receiver r, capacity c_a on each arc a
Solution	A feasible flow $x \in \mathcal{X}_{\mathcal{P}}$ in D
Objective	Maximize the residual throughput $\rho(x)$ of flow x as defined in Equation (6.4)

We denote by ρ_F the value of a maximum survivable flow for the considered instance:

$$\rho_F = \max_{x \in \mathcal{X}_{\mathcal{P}}} \rho(x) \tag{6.5}$$

Notice that a maximum survivable flow x is *static*: one has to decide how to route flow before knowing which arc will eventually fail.

6.1.3.2 Linear programming formulations

The maximum survivable flow problem can be modelled using linear programming:

$$\begin{cases} \rho_F = \max \rho & (6.6) \\ \text{s.t.} \quad \rho \leq \displaystyle\sum_{p \in \mathcal{P} \setminus \mathcal{P}_a} x_p \quad \forall a \in A & (6.7) \\ \displaystyle\sum_{p \in \mathcal{P}_a} x_p \leq c_a \quad \forall a \in A & (6.8) \\ x_p \geq 0 \quad \forall p \in \mathcal{P} & (6.9) \\ & (6.10) \end{cases}$$

For each simple path p, variable x_p stands for the amount of flow conveyed by p. Hence, the number of variables in the above linear program may be exponential in the size of the instance. Constraints (6.7) along with the maximization of variable ρ ensure that a residual throughput of value ρ is experienced by the receiver regardless of whether any arc of the network fails. Finally, Constraints (6.8) are the classical capacity requirements. From this *path formulation* of the maximum survivable flow problem it is possible to deduce an *arc*

formulation of the same problem to which is associated the linear program below, proposed by Aneja et al. in [129]:

$$\begin{cases} \rho_F = \max \rho & (6.11) \\ \text{s.t.} \quad \rho + f_a \leq \phi \quad \forall a \in A & (6.12) \\ \quad \sum_{a \in \delta^+(v)} f_a - \sum_{a \in \delta^-(v)} f_a = b_v(\phi) \quad \forall v \in V & (6.13) \\ \quad f_a \leq c_a \quad \forall a \in A & (6.14) \\ \quad \phi, f_a \geq 0 \quad \forall a \in A & (6.15) \end{cases}$$

where

$$b_v(\phi) = \begin{cases} \phi & \text{if } v = s \quad (6.16) \\ -\phi & \text{if } v = r \quad (6.17) \\ 0 & \text{otherwise} \quad (6.18) \end{cases}$$

For each arc a, variable f_a stands for the amount of flow going through a, whereas variable ϕ and ρ respectively model the *nominal* and the *residual* throughput experienced by receiver r.

6.1.3.3 Complexity and algorithms

This problem has been thoroughly studied by Aneja et al. in [129]. They provide an algorithm, based on Newton's method, which returns a maximum survivable flow by performing $O(n)$ calls to a subroutine solving a maximum flow problem in a network obtained from the original one by suitably reducing the value of each capacity in the graph. Hence this problem can be solved in strongly polynomial time.

The authors of [129] also show that the maximum survivable flow problem has some nice properties. Firstly, given a maximum survivable flow, either it saturates each capacity of the arcs of a minimum cut when no failure occurs, or there exists an augmenting path between the source and the receiver in the residual network, and routing flow along that path increases the value of the nominal throughput. Furthermore, the new flow thereby obtained is still optimal with respect to the maximum survivable flow problem. Hence, it is possible to obtain in strongly polynomial time a maximum survivable flow which is also a maximum flow (without considering failures). This means there is no need to find a

trade-off between the nominal throughput and the survivable one as both criteria can be simultaneously optimized.

Another remarkable feature of this problem arises when one considers its integral version (by requiring the flow on each path to be a non-negative integer). Although the continuous relaxation may not provide an integer flow, it is possible to get an integral maximum survivable flow, starting from a fractional one, by performing two more maximum flow computations. See [129] for further details.

We would like to mention that an extension of the maximum survivable flow problem, with an arbitrary number of failures, is studied by Bertsimas et al. in [130, 131] under the name *robust maximum flow problem*. In [131], a nice interpretation of this class of survivable flow problems is given in the light of game theory: the maximum survivable flow problem can be regarded as a game between a flow player and an adversarial interdictor. The flow player first picks a set of paths on which flow will be routed, then he decides how much flow should be conveyed by each path, so as to maximize the residual throughput obtained once the interdictor, who plays in second position, will have removed the most *critical arc* of the network, namely the arc whose deletion incurs the greatest loss in terms of throughput value for the flow player.

6.2 Maximum survivable Steiner flow

6.2.1 Problem statement

We will now study the extension of the previous problem to a multicast network. As before, we focus on the case of a directed network since it encompasses the case of an undirected one. As in the previous chapter, we will make a distinction between a directed network (which may be bidirected) prone to *single arc failures* and a bidirected network prone to *single link failures*. In the latter case we assume that *both arcs* of the link are simultaneously failing while in the former case only one arc fails. The instance is made of a network $D = (V, A)$ with a *source* s, a set R of *receivers*, and a capacity c_a for each arc a. A *Steiner flow* x is an assignment of a non-negative real value x_t to each Steiner tree t spanning s and R. A Steiner flow x is *feasible* if it satisfies the capacity constraints. We

denote by $\mathcal{X}_\mathcal{T}$ the set of all feasible Steiner flows:

$$\mathcal{X}_\mathcal{T} = \left\{ x \in \mathbb{R}_+^\mathcal{T} : \sum_{t \in \mathcal{T}_a} x_t \leq c_a \; \forall a \in A \right\} \tag{6.19}$$

where $\mathcal{T}$ is the set of all Steiner trees spanning s and R, while $\mathcal{T}_a$ is the subset of all trees in $\mathcal{T}$ using arc a. Given a Steiner tree t and a receiver r, we denote by p_t^r the unique path from s to r in t. For each arc a and each receiver r, let $\mathcal{T}_a^r$ be the set of all Steiner trees t using a to convey flow toward r, namely $a \in p_t^r$.

We assume again that any arc a of the network may fail, and once again, we model this event by removing the corresponding arc from the network. As in the previous chapter, the impact of the failure of arc a on a given feasible Steiner flow x differs from the impact of the same failure on a classical flow. Indeed, we model this impact by preventing a Steiner tree t using a from conveying flow between the source and any receiver r such that the path p_t^r contains arc a. Hence, a receiver lying in the same connected component (which is a sub-tree of t) as the source after removal of the failing arc will still receive the flow conveyed by t. Figure 5.1 provides an example (see the previous chapter). Given a feasible Steiner flow x, we define, for each arc a and each receiver r, the *residual throughput* $\rho_a^r(x)$ *of* x *with respect to receiver* r*, whenever arc* a *fails*, as the quantity:

$$\rho_a^r(x) = \sum_{t \in \mathcal{T}} x_t - \sum_{t \in \mathcal{T}_a^r} x_t \tag{6.20}$$

or equivalently:

$$\rho_a^r(x) = \sum_{t \in \mathcal{T} \setminus \mathcal{T}_a^r} x_t \tag{6.21}$$

Observe that the first term in the right-hand side of Equation (6.20) is the *nominal throughput*, the amount of flow which can be conveyed to each receiver when no failure occurs, while the second term (in the right-hand side of the same equation) is the fraction of flow on arc a routed toward receiver r. Hence $\rho_a^r(x)$ is exactly the amount of flow received by the receiver r when arc a fails. The *residual throughput* $\rho(x)$ *of a given Steiner flow* x is defined as the minimum, over all receivers and all possible arc failures, of the residual throughput $\rho_a^r(x)$ associated to receiver r when arc a fails:

$$\rho(x) = \min_{r \in R} \min_{a \in A} \rho_a^r(x) \tag{6.22}$$

We are now ready to define formally our problem:

Problem	*Maximum survivable Steiner flow*
Instance	Network $D = (V, A)$, source s, set of receivers R, capacity c_a on each arc a
Solution	A feasible Steiner flow $x \in \mathcal{X}_{\mathcal{T}}$ in D
Objective	Maximize the residual throughput $\rho(x)$ of Steiner flow x as defined in Equation (6.22)

We denote by ρ_S the value of a maximum survivable Steiner flow for the considered instance:

$$\rho_S = \max_{x \in \mathcal{X}_{\mathcal{T}}} \rho(x) \tag{6.23}$$

Observe that a maximum survivable Steiner flow x is *static*. When the set R is a singleton, a Steiner tree spanning s and R is a simple path between the source and the single receiver. Hence the maximum survivable Steiner flow problem can be regarded as a generalization of the maximum survivable flow problem to more than one receivers.

6.2.2 A detailed example

We keep working with the directed butterfly network which is depicted on Figure 6.1 a). We are looking for a maximum survivable Steiner flow in the instance where the capacity of each arc a of the directed butterfly network is set to $c_a = 1$. Such a maximum survivable Steiner flow $\Join$ is depicted on Figure 6.1 b). Steiner flow $\Join$ uses two trees, each one routing half a unit of flow. Table 6.1 provides, for each arc a and each receiver r, the residual throughput $\rho_a^r(\Join)$ experienced by r when a fails. By definition, the residual throughput $\rho(\Join)$ is the minimum value which appears in Table 6.1, namely $\rho(\Join) = \frac{1}{2}$.

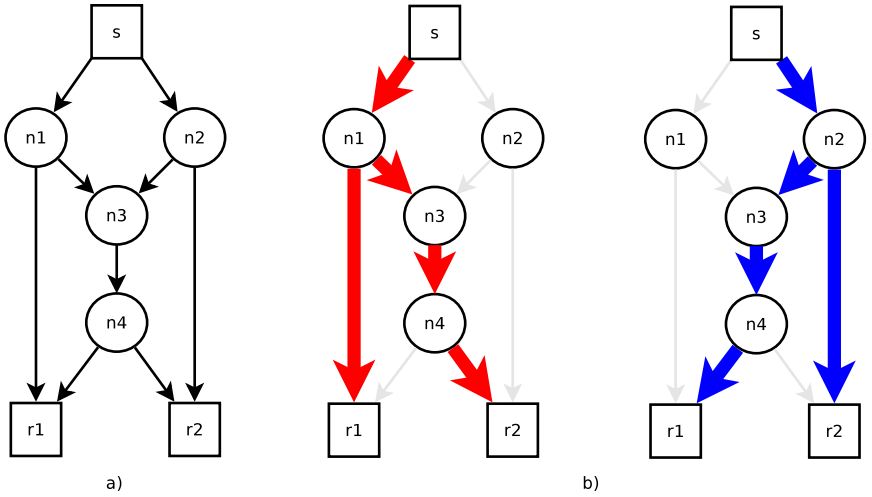


Figure 6.1 – a) The directed butterfly network. b) A maximum survivable Steiner flow using two trees in the directed butterfly network. Each tree conveys half a unit of flow, so that the residual throughput of the induced Steiner flow is $\frac{1}{2}$.

Failure	r_1	r_2
(s, n_1)	1/2	1/2
(s, n_2)	1/2	1/2
(n_1, r_1)	1/2	1
(n_1, n_3)	1	1/2
(n_2, r_2)	1	1/2
(n_2, n_3)	1/2	1
(n_3, n_4)	1/2	1/2
(n_4, r_1)	1/2	1
(n_4, r_2)	1	1/2

Table 6.1 – Impact of the failure of each arc on the residual throughput experienced by each receiver.

6.2.3 Linear programming formulation

The maximum survivable Steiner flow problem can be formalized using linear programming:

$$\begin{cases} \rho_S = \max \quad \rho & (6.24) \\ \text{s.t.} \quad \rho \leq \displaystyle\sum_{t \in \mathcal{T} \setminus \mathcal{T}_a^r} x_t \quad \forall a \in A, r \in R & (6.25) \\ \displaystyle\sum_{t \in \mathcal{T}_a} x_t \leq c_a \quad \forall a \in A & (6.26) \\ x_t \geq 0 \quad \forall t \in \mathcal{T} & (6.27) \\ & (6.28) \end{cases}$$

For each Steiner tree t, variable x_t stands for the amount of flow conveyed by t. Hence, the number of variables in the above linear program may be exponential in the size of the instance. Constraints (6.25) along with the maximization of variable ρ ensure that, for any Steiner flow x satisfying the capacity requirements modeled by Constraints (6.26), each receiver experiences a throughput of value at least ρ regardless of whether any arc a fails. If one is interested by the model of *single link failures* in a bidirected network $B = (V, L)$, Constraints (6.25) should be replaced by the following ones:

$$\rho \leq \sum_{t \in \mathcal{T}} x_t - \left(\sum_{t \in \mathcal{T}_{a_1}^r} x_t + \sum_{t \in \mathcal{T}_{a_2}^r} x_t \right) \quad \forall \ell = \{a_1, a_2\} \in L, r \in R \tag{6.29}$$

where we rest on the fact that a Steiner tree is using at most *one* arc of each link, so that the two sets $\mathcal{T}_{a_1}^r$ and $\mathcal{T}_{a_2}^r$ are disjoint for any link $\ell = \{a_1, a_2\}$ and any receiver r. We denote by y the vector of dual variables associated to Constraints (6.26), and by z the one corresponding to Constraints (6.25). The dual of the previous linear program is:

$$\begin{cases} \rho_S = \min \quad \displaystyle\sum_{a \in A} c_a y_a & (6.30) \\ \text{s.t.} \quad \displaystyle\sum_{a \in A} \sum_{r \in R} z_a^r = 1 & (6.31) \\ \displaystyle\sum_{a \in t} y_a + \sum_{r \in R} \sum_{a \in p_t^r} z_a^r \geq 1 \quad \forall t \in \mathcal{T} & (6.32) \\ y_a, z_a^r \geq 0 \quad \forall a \in A, r \in R & (6.33) \end{cases}$$

where we get Constraints (6.32) by combining Constraint (6.31) with the set of dual constraints associated to the primal vector of variables x:

$$\sum_{a \in t} y_a \geq \sum_{a \in A} \sum_{r \in R} z_a^r - \sum_{r \in R} \sum_{a \in p_t^r} z_a^r \quad \forall t \in \mathcal{T} \tag{6.34}$$

6.2.4 Complexity and algorithms

Observe that the pricing problem is exactly the same as the one appearing in the study of the minimum cost survivable Steiner flow problem in the previous chapter. We hence know from Theorem 24 and Theorem 25 that this pricing problem is NP-hard in both directed and undirected networks. However the coefficients in the objective function of the dual of the present problem are not arbitrary since the coefficient associated to variable z_a^r equals 0 for each arc a and each receiver r. It is thus impossible to apply the theorem established by Grötschel, Lovász, and Schrijver on the equivalence between polynomial-time separation and polynomial-time optimization for convex polytopes [90]. Again, the previous discussion only emphasizes that, unless $P = NP$, it is impossible to devise an algorithm running in polynomial time by combining the ellipsoid method with an oracle solving the pricing problem. We acknowledge that it may still be possible to find an algorithm which would solve the maximum survivable Steiner flow problem in polynomial time (for instance, by using another linear programming formulation). We again refer the interested reader to the paper by Nace et al. [123] for a non-trivial example of a problem whose extended linear programming formulation admits an NP-hard separation problem while the problem itself can be solved in polynomial time. Hence, we shall leave the complexity of finding a maximum survivable Steiner flow as an open question:

Conjecture 2. *We believe that the maximum survivable Steiner flow problem is NP-hard.*

6.2.5 Computing a maximum survivable Steiner flow by column generation

One can solve the maximum survivable Steiner flow problem by using column generation techniques. Given dual vectors (y, z), computing the *reduced cost* σ_t associated to variable x_t (which we will refer to as the reduced cost of tree t by a slight abuse of language) amounts

to solving the separation problem associated to Constraints (6.32). More precisely:

$$\sigma_t = 1 - \left(\sum_{a \in t} y_a + \sum_{r \in R} \sum_{a \in p_t^r} z_a^r \right) \tag{6.35}$$

so that one is looking for a tree whose cost, with respect to the weights vectors y and z, is minimum. It is possible to solve the pricing problem to optimality by using the mixed-integer linear programming formulation presented in the previous chapter. If, by solving this problem, we find a tree with a positive reduced cost, this tree represents a new column to be added to the master problem, which is then solved again. If we are not able to exhibit such a tree, then it means that we have obtained an optimal solution.

6.2.6 Nominal and residual throughputs

We study the small network depicted on Figure 6.2 a). This network is made of one source s, two receivers r_1 and r_2, along with two intermediate nodes u and v. The capacity of each arc is set to 1 except for arc (u, v) whose capacity is 2. Consider the Steiner flow x depicted on Figure 6.2 b) which is supported by two trees, each one routing 1 unit of flow, so that each receiver gets the two messages a and b. Observe that x is a maximum survivable Steiner flow with a residual throughput $\rho(x) = 1$ and a nominal throughput of 2 *according to the model*. However, we claim that it is *not* possible to simultaneously route both messages to each receiver while ensuring that regardless of which arc may fail, both receivers will get the *same* content. To see why, notice that if arc (u, v) fails, Steiner flow x can not deliver the same message to both receivers although $\rho(x) = 1$. For this reason, we disregard the nominal throughput predicted by our model and we decide that the throughput experienced by each receiver is ρ_S regardless of whether a failure actually occurs or not, as suggested by Figure 6.2 c).

6.3 Maximum survivable coded flow

6.3.1 Problem statement

This time, we will distinguish among the three cases of bidirected, directed, and undirected networks. We first focus on the case of directed networks. The setting is exactly

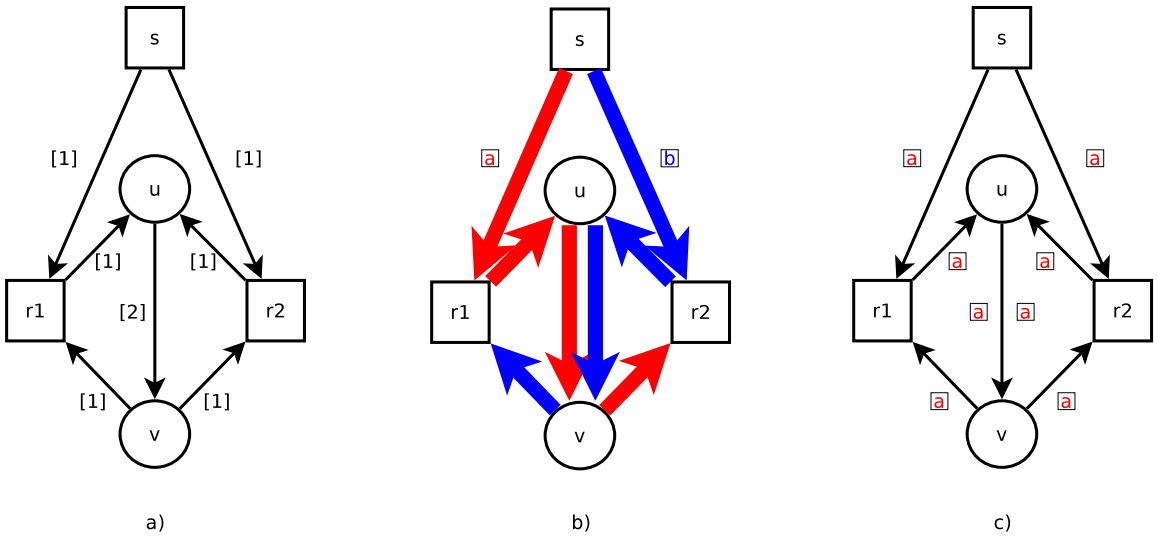


Figure 6.2 – a) A small network with one source s and two receivers r_1 and r_2. The capacity of each arc is set to 1 except for the arc (u, v) whose capacity is 2. b) A Steiner flow supported by two trees, each tree conveys 1 unit of flow, so that each receiver get the two messages a and b. c) In order to guarantee that both receivers get the same content when arc (u, v) fails, it is required to send twice the same message.

the same as in the previous problem: The instance is made of a network $D = (V, A)$ with a *source* s, a set R of *receivers*, and a capacity c_a for each arc a. Recall that a coded flow x is an assignment of a non-negative real value x_p to each simple path p from the source to any receiver r. A coded flow x is *feasible* if it satisfies the network coding capacity constraints. We denote by $\mathcal{X}_\mathcal{P}$ the set of all feasible coded flows

$$\mathcal{X}_\mathcal{P} = \left\{ x \in \mathbb{R}_+^\mathcal{P} : \sum_{p \in \mathcal{P}_a^r} x_t \leq c_a \, \forall a \in A, r \in R \right\} \tag{6.36}$$

where for any receiver r, $\mathcal{P}^r$ is the set of all simple paths from s to r, while $\mathcal{P}_a^r$ is the subset of all paths in $\mathcal{P}^r$ using arc a. We also denote by $\mathcal{P}$ the set of all simple paths between the source and any receiver, namely $\mathcal{P} = \bigcup_{r \in R} \mathcal{P}^r$.

We assume again that any arc a of the network may fail, which we still model by removing the corresponding arc from the graph. As in the case of the maximum survivable flow problem, the failure of arc a prevents any path, connecting the source to a receiver thanks to a, to route flow to this receiver. This means a coded flow x can only use paths not affected by the failure to actually convey flow. Given a feasible coded flow x, we define, for each arc a and each receiver r, the *residual throughput* $\rho_a^r(x)$ *of* x *with respect to receiver*

r, whenever arc a fails, as the quantity:

$$\rho_a^r(x) = \sum_{p \in \mathcal{P}^r} x_p - \sum_{p \in \mathcal{P}_a^r} x_p \tag{6.37}$$

or equivalently:

$$\rho_a^r(x) = \sum_{p \in \mathcal{P}^r \setminus \mathcal{P}_a^r} x_p \tag{6.38}$$

Observe that the first term in the right-hand side of Equation (6.37) is the *nominal throughput* for receiver r, the amount of flow which can be conveyed to this receiver when no failure occurs, while the second term (in the right-hand side of the same equation) is the fraction of flow on arc a routed toward r. Thus, $\rho_a^r(x)$ is exactly the throughput experienced by receiver r when arc a fails. The *residual throughput $\rho(x)$ of a given coded flow x* is defined as the minimum, over all receivers and all possible arc failures, of the residual throughput $\rho_a^r(x)$ induced for receiver r by the failure of arc a:

$$\rho(x) = \min_{r \in R} \min_{a \in A} \rho_a^r(x) \tag{6.39}$$

We are now ready to define formally our problem:

Problem	*Maximum survivable coded flow*
Instance	Network $D = (V, A)$, source s, set of receivers R, capacity c_a on each arc a
Solution	A feasible coded flow $x \in \mathcal{X}_\mathcal{P}$ in D
Objective	Maximize the residual throughput $\rho(x)$ of coded flow x as defined in Equation (6.39)

We denote by ρ_C the value of a maximum survivable coded flow for the considered instance:

$$\rho_C = \max_{x \in \mathcal{X}_\mathcal{P}} \rho(x) \tag{6.40}$$

Notice that a maximum survivable coded flow x is *static*. This underlies the need for a static coding scheme, common to all failure patterns. We shall address this issue later. When the set R is a singleton, the set of paths $\mathcal{P}$ is exactly the set of all simple paths from the source to the single receiver. Hence the maximum survivable coded flow problem can also be regarded as a generalization of the maximum survivable flow problem to multiple receivers.

6.3.2 Example continued

Consider again the instance made of the directed butterfly network depicted on Figure 6.1 a) with capacity $c_a = 1$ for each arc a. A maximum survivable coded flow $\varkappa$ for this instance is depicted on Figure 6.3. Coded flow $\varkappa$ uses two paths routing one unit of flow to convey the data from the source to each receiver, so that each receiver experiences a throughput of value 1 regardless of whether any arc is removed from the network. Thus the residual throughput of coded flow $\varkappa$ is $\rho(\varkappa) = 1$.

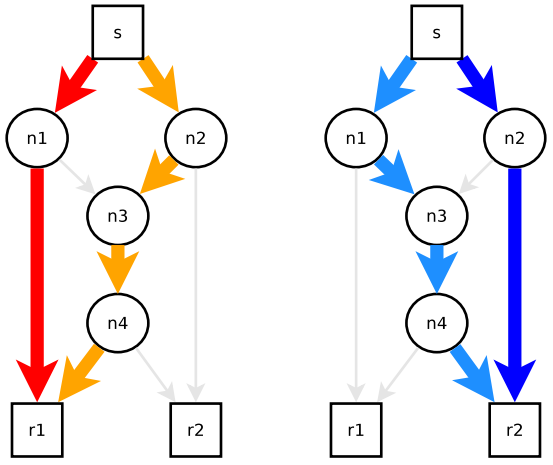


Figure 6.3 – A maximum survivable coded flow $\varkappa$ using exactly two paths to route the data from the source to each receiver. Each path is routing one unit of flow, so that the residual throughput of coded flow $\varkappa$ is $\rho(\varkappa) = 1$.

6.3.3 Linear programming formulations

6.3.3.1 Directed and bidirected networks

We shall now provide a linear programming formulation of the maximum survivable coded flow problem in a directed network:

$$\begin{cases} \rho_C = \max \quad \rho & (6.41) \\ \text{s.t.} \quad \rho \leq \displaystyle\sum_{p \in \mathcal{P}^r \setminus \mathcal{P}^r_a} x_p \quad \forall a \in A, r \in R & (6.42) \\ \displaystyle\sum_{p \in \mathcal{P}^r_a} x_p \leq c_a \quad \forall a \in A, r \in R & (6.43) \\ x_p \geq 0 \quad \forall p \in \mathcal{P} & (6.44) \\ & (6.45) \end{cases}$$

For each simple path p, variable x_p stands for the amount of flow conveyed by p. Hence, the number of variables in the above linear program may again be exponential in the size of the instance. Constraints (6.42) along with the maximization of variable ρ ensure that, for any coded flow x satisfying the capacity requirements modeled by Constraints (6.43), each receiver experiences a throughput of value at least ρ regardless of whether any arc a fails. If one is interested by the model of *single link failures* in a bidirected network $B = (V, L)$, Constraints (6.42) should be replaced by the following ones:

$$\rho \leq \sum_{p \in \mathcal{P}^r} x_p - \left(\sum_{p \in \mathcal{P}^r_{a_1}} x_p + \sum_{p \in \mathcal{P}^r_{a_2}} x_p \right) \quad \forall \ell = \{a_1, a_2\} \in L, r \in R \tag{6.46}$$

where we rest on the fact that a simple path is using at most *one* arc of each link, so that the two sets $\mathcal{P}^r_{a_1}$ and $\mathcal{P}^r_{a_2}$ are disjoint for any link $\ell = \{a_1, a_2\}$ and any receiver r.

The previous linear program allows to solve the *path formulation* of the maximum survivable coded flow problem. It is also interesting to consider the linear program below which is associated to the *arc formulation* of the same problem:

$$\begin{cases} \rho_C = \max \quad \rho & (6.47) \\ \text{s.t.} \quad \rho \leq \phi^r - f^r_a \quad \forall a \in A, r \in R & (6.48) \\ \quad \sum_{a \in \delta^+(v)} f^r_a - \sum_{a \in \delta^-(v)} f^r_a = b^r_v(\phi^r) \quad \forall r \in R, v \in V & (6.49) \\ \quad f^r_a \leq c_a \quad \forall a \in A, r \in R & (6.50) \\ \quad \phi^r, f^r_a \geq 0 \quad \forall a \in A, r \in R & (6.51) \end{cases}$$

with

$$b^r_v(\phi) = \begin{cases} \phi & \text{if } v = s \quad (6.52) \\ -\phi & \text{if } v = r \quad (6.53) \\ 0 & \text{otherwise} \quad (6.54) \end{cases}$$

For each arc a and each receiver r, variable f^r_a stands for the amount of flow going from source s to receiver r through a, while variable ϕ^r models the throughput experienced by receiver r.

6.3.3.2 Undirected networks

The linear program below allows to *define* and *compute* a maximum survivable coded flow in an undirected network $G = (V, E)$ with source s, set of receivers R, and capacity

c_e for each edge e. The idea is the same as the one used to find a maximum coded flow in an undirected network: Consider the bidirected network $B = (V, L)$ obtained by replacing each edge of G by a *pair of reverse arcs*. Keep the source and the receivers as defined in G. Finally, assign a *variable capacity* h_a on each arc a of the bidirected network then add an *orientation constraint* for each link ℓ. The following linear program allows to compute a maximum survivable coded flow over all possible orientations of the undirected network:

$$
\begin{cases}
\rho_C = \max \quad \rho & (6.55) \\
\text{s.t.} \quad \rho \leq \sum_{p \in \mathcal{P}^r} x_p - \left(\sum_{p \in \mathcal{P}^r_{a_1}} x_p + \sum_{p \in \mathcal{P}^r_{a_2}} x_p \right) \quad \forall \ell = \{a_1, a_2\} \in L, r \in R & (6.56) \\
\quad \sum_{p \in \mathcal{P}^r_a} x_p \leq h_a \quad \forall a \in \ell, \ell \in L, r \in R & (6.57) \\
\quad h_{a_1} + h_{a_2} \leq c_\ell \quad \forall \ell = \{a_1, a_2\} \in L & (6.58) \\
\quad h_a, x_p \geq 0 \quad \forall a \in \ell, \ell \in L, p \in \mathcal{P} & (6.59) \\
& (6.60)
\end{cases}
$$

For each simple path p, variable x_p stands again for the amount of flow conveyed by p, while, for each arc a, variable h_a models the amount of *coded* flow going through a. Notice that this linear program may also have an exponential number of variables with respect to the instance size. Here also, Constraints (6.56) along with the maximization of variable ρ ensure that, for any coded flow x, the throughput experienced by any receiver whenever any arc a fails is at least ρ. Constraints (6.57) are capacity requirements with a *variable capacity* h_a for each arc a. Finally, Constraints (6.58) are the previously mentioned *orientation constraints* which induce a *flow coupling* between all receivers. Instead of dealing with the previous *path formulation* of the maximum survivable coded flow problem, it is often more convenient to work with the linear program below, which is associated to the *arc*

formulation of the present problem:

$$\begin{cases} \rho_C = \max \quad \rho & (6.61) \\ \text{s.t.} \quad \rho + f_a^r \leq \phi^r \quad \forall a \in A, r \in R & (6.62) \\ \quad \sum_{a \in \delta^+(v)} f_a^r - \sum_{a \in \delta^-(v)} f_a^r = b_v^r(\phi^r) \quad \forall r \in R, v \in V & (6.63) \\ \quad f_a^r \leq h_a \quad \forall a \in \ell, \ell \in L, r \in R & (6.64) \\ \quad h_{a_1} + h_{a_2} \leq c_\ell \quad \forall \ell = \{a_1, a_2\} \in L & (6.65) \\ \quad \phi^r, f_a^r, h_a \geq 0 \quad \forall a \in \ell, \ell \in L, r \in R & (6.66) \\ & (6.67) \end{cases}$$

with

$$b_v^r(\varphi) = \begin{cases} \varphi & \text{if } v = s \quad (6.68) \\ -\varphi & \text{if } v = r \quad (6.69) \\ 0 & \text{otherwise} \quad (6.70) \end{cases}$$

For each arc a, and each receiver r of the bidirected network $B = (V, L)$, variable f_a^r models the amount of flow going from s to r through a (as in the case of a directed network). Again, Constraints (6.62) along with along with the maximization of variable ρ ensure that, for any coded flow x, the throughput experienced by any receiver whenever any arc a fails is at least ρ. The other variables and constraints are similar in both formulations.

6.3.4 Complexity and algorithms

Regardless of the network structure, the linear program obtained by considering the arc formulation of the maximum survivable coded flow problem involves a polynomial number of (continuous) variables and constraints (with respect to the instance size). Hence, it is possible to compute an arc decomposition of a maximum survivable coded flow in polynomial time. Given such an arc decomposition f, a path decomposition x can be built in polynomial time thanks to the technique presented in the proof of Lemma 1: For each receiver r, let f^r be the vector of components f_a^r over each arc a of the network. Observe that vector f^r is the arc projection of a *simple* flow between the source and receiver r. For each receiver r, compute a path decomposition of this simple flow using the algorithm presented by Ahuja et al. [7, Theorem 3.5] which runs in polynomial time. The

concatenation of all those path decompositions (one per receiver) is clearly a valid path decomposition of the coded flow.

The next theorem provides a connection between the maximum survivable coded flow problem and the maximum survivable flow problem in a directed network $D = (V, A)$:

Theorem 29. *In a directed network $D = (V, A)$, the maximum survivable coded flow problem can be decomposed into a set of instances of the maximum survivable flow problem, one per receiver, and $\rho_C = \min_{r \in R} \rho_F^r$ where for each receiver r, ρ_F^r is the value of a maximum survivable flow in the instance induced by r.*

Proof. Starting from the above linear programming formulation, we introduce a new variable ρ^r for each receiver r and we replace Constraints (6.42) by the following constraints for each receiver r

$$\rho \leq \rho^r \quad \text{and} \quad \rho^r \leq \sum_{p \in \mathcal{P}^r} x_p - \sum_{p \in \mathcal{P}_a^r} x_p \quad \forall a \in A \tag{6.71}$$

Observe that we thus obtain a set of completely independent problems, one for each receiver r. Furthermore, each one is exactly a maximum survivable flow problem between the source and receiver r. □

This decomposition combined with the algorithm proposed by Aneja et al. in [129] immediately imply the following result:

Corollary 13. *In a directed network $D = (V, A)$, the maximum survivable coded flow problem can be solved in strongly polynomial time by $O(|R||V|)$ calls to a subroutine solving the maximum flow problem.*

In an undirected network, the coupling induced by the orientation constraints precludes such a decomposition.

6.3.5 Coding scheme

In a directed acyclic network, there exists a *static coding scheme common to all failure patterns* ensuring residual throughput ρ_C thanks to [59, Theorems 11]. We refer the reader

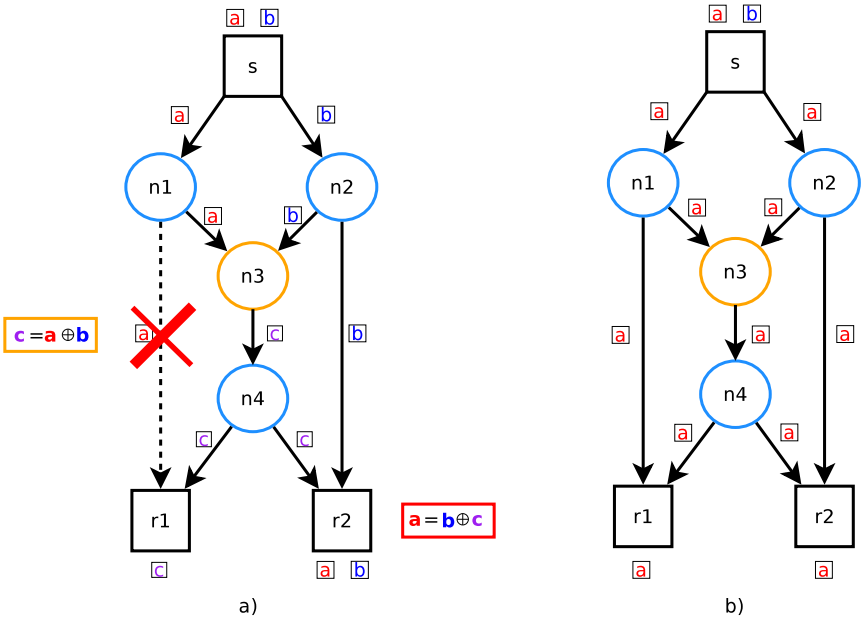


Figure 6.4 – a) A coding scheme associated to a maximum survivable coded flow in the directed butterfly network. Unfortunately, receiver r_1 can not decode the coded message c when the arc (n_1, r_1) fails. b) A coding scheme ensuring that each receiver gets message a regardless of whether an arc fails or not.

to the discussion in the previous chapter regarding the design of a coding scheme for a minimum-cost survivable coded flow. In networks with cycles, it is still possible to design a fault-tolerant coding scheme by using more complicated coding techniques, as explained in [59].

6.3.6 Nominal and residual throughputs

According to our model there exists a maximum survivable coded flow x in the directed butterfly network with a residual throughput $\rho(x) = 1$ and a nominal throughput of 2. However, consider the coding scheme depicted on Figure 6.4 a) and observe that whenever the arc (n_1, r_1) fails, receiver r_1 only gets the coded message c which can *not* be decoded. For this reason, we disregard the nominal throughput predicted by our model and we decide that the throughput experienced by each receiver is ρ_C regardless of whether a failure actually occurs or not, as suggested by Figure 6.4 b).

6.4 Features of the information flows

6.4.1 Survivable coding gain

We shall now study the impact of implementing coding techniques in the present setting. We are naturally led to ask the following question:

Question 5. *Is there an incentive, with respect to the residual throughput, to use network coding in a multicast network prone to single link failures?*

Similarly to the previously studied frameworks, we introduce the notion below so as to facilitate comparisons:

Definition 6. *Given a network $D = (V, A)$ with source s, set of receivers R, and capacity c_a for each arc a, the* survivable coding gain *g_ρ, provided by network coding over multicast alone, is the ratio of the value of a maximum survivable coded flow over the one of a maximum survivable Steiner flow:*

$$g_\rho = \frac{\rho_C}{\rho_S} \tag{6.72}$$

Recall that we only consider networks where there is at least two arc-disjoint paths from the source to any receiver, so that the quantity ρ_S is positive and the previous definition of the *survivable coding gain* g_ρ is well-grounded. The next theorem shows that adding the possibility to perform coding operations in a multicast network never impedes the performance of this network with respect to the residual throughput:

Theorem 30. *Given a network $D = (V, A)$ with source s, set of receivers R, and capacity c_a for each arc a, the survivable coding gain g_ρ is at least* 1*.*

The proof of Theorem 30 is very similar to the one of Theorem 3 and can be found in Appendix C.1.5. Notice that, with respect to the residual throughput, there is an incentive to use network coding only if $g_\rho > 1$. We shall now study the survivable coding gain of some networks.

6.4.2 Features of the survivable coding gain

6.4.2.1 Single receiver

We first give a word about the case where there is only one single receiver. In this case both the maximum survivable Steiner flow problem and the maximum survivable coded flow problem become equivalent to the maximum survivable flow problem, as emphasized by the next lemma:

Lemma 15. *Given a network $D = (V, A)$ with source s, a single receiver r distinct from s, and capacity c_a for each arc a, the survivable coding gain g_ρ equals* 1.

6.4.2.2 Survivable coding gain and survivable cost coding gain

The following theorem establishes a formal connection between two a priori unrelated quantities, namely the survivable coding gain g_ρ and the survivable cost coding gain g_χ. This result can be regarded as an extension of [79, Theorem 9] to the present survivable setting.

Theorem 31. *Given a network $D = (V, A)$ with source s and set of receivers R, denote by $g_\rho(D, c)$ the* survivable coding gain *obtained by setting capacity c_a on each arc a. Also denote by $g_\chi(D, w)$ the* survivable cost coding gain *obtained by setting non-negative weight w_a on each arc a along with positive demand d. Then,*

$$\max_{c \geq 0} g_\rho(D, c) = \max_{w \geq 0} g_\chi(D, w) \tag{6.73}$$

Proof. The survivable cost coding gain g_χ is left unchanged after scaling of the weight vector by a positive factor α, namely $g_\chi(D, \alpha w) = g_\chi(D, w)$. This invariance of g_χ also holds if the demand is scaled by a positive factor, so that we can assume $d = 1$ without loss of generality. Similarly, scaling the capacity vector by a positive factor α will leave the survivable coding gain g_ρ unaltered, namely $g_\rho(D, \alpha c) = g_\rho(D, c)$. We denote by $\chi_S(D, w)$, respectively $\chi_C(D, w)$, the cost of a minimum-cost survivable Steiner, respectively coded, flow for the instance induced by a given weight vector w, so that $g_\chi(D, w) = \frac{\chi_S(D,w)}{\chi_C(D,w)}$. Similarly,

let $\rho_S(D,c)$, respectively $\rho_C(D,c)$, be the residual throughput of a maximum survivable Steiner, respectively coded, flow for the instance induced by a given capacity vector c, which means that $g_\rho(D,c) = \frac{\rho_C(D,c)}{\rho_S(D,c)}$. We first prove the inequality $\max_{c\geq 0} g_\rho(D,c) \leq \max_{w\geq 0} g_\chi(D,w)$. Let c be a capacity vector such that $\rho_C(D,c) > 0$. Without loss of generality, we can assume $\rho_C(D,c) = 1$ by scaling capacity vector c by $[\rho_C(D,c)]^{-1}$. Let $\mathcal{x}$ be a maximum survivable coded flow with respect to capacity vector c, namely $\rho(\mathcal{x}) = \rho_C(D,c)$. Let also $(\mathbb{y},\mathbb{z})$ be an optimal solution to the dual of the maximum survivable Steiner flow problem. Since $\rho_C(D,c) = 1$ and $d = 1$, the couple of vectors $(\mathcal{x},c)$ is a feasible solution to the minimum-cost survivable coded flow problem for the instance induced by weight vector $\mathbb{y}$ (where the weight of each arc a is set to $\mathbb{y}_a$). Furthermore, the cost of this feasible solution is $\sum_{a\in A} \mathbb{y}_a c_a$ which is exactly $\rho_S(D,c)$ by optimality of $(\mathbb{y},\mathbb{z})$. This immediately implies:

$$\chi_C(D,\mathbb{y}) \leq \rho_S(D,c) \tag{6.74}$$

Moreover, by feasibility of $(\mathbb{y},\mathbb{z})$ (with respect to the dual of the maximum survivable Steiner flow problem), vector $\mathbb{z}$ is a feasible solution to the dual of the minimum cost survivable Steiner flow problem for the instance induced by weight vector $\mathbb{y}$. Furthermore, since $d = 1$ the value of this solution is $\sum_{a\in A}\sum_{r\in R} \mathbb{z}_a^r$ which equals 1 by definition of $(\mathbb{y},\mathbb{z})$. Combining this last result with our assumption that $\rho_C(D,c) = 1$, we obtain:

$$\rho_C(D,c) \leq \chi_S(D,\mathbb{y}) \tag{6.75}$$

Combining Equation (6.74) and Equation (6.75), we deduce that for any capacity vector c, we can find a weight vector $\mathbb{y}$ which satisfies:

$$g_\rho(D,c) \leq g_\chi(D,\mathbb{y}) \tag{6.76}$$

which directly implies that $\max_{c\geq 0} g_\rho(D,c) \leq \max_{w\geq 0} g_\chi(D,w)$. Conversely, we shall prove the inequality $\max_{w\geq 0} g_\chi(D,w) \leq \max_{c\geq 0} g_\rho(D,c)$. Let w be a weight vector such that $\chi_S(D,w) > 0$. Without loss of generality, we can assume $\chi_S(D,w) = 1$ by scaling weight vector w by $[\chi_S(D,w)]^{-1}$. Let $\mathcal{x}$ be a minimum-cost survivable coded flow with respect to weight vector w, under the assumption $d = 1$, namely $\rho(\mathcal{x}) = \rho_C(D,c)$. Let $\hbar$ be the coding vector associated to coded flow $\mathcal{x}$, namely $\hbar_a = \max_{r\in R} \sum_{p\in\mathcal{P}_a^r} \mathcal{x}_p$. It should be clear

that coded flow χ is a feasible solution to the maximum survivable coded flow problem for the instance induced by capacity vector $\hbar$ (where the capacity of each arc a is set to $\hbar_a$). Furthermore, its residual throughput $\rho(\chi)$ is at least $d = 1$. Combining this last result with our assumption that $\chi_S(D, w) = 1$, we get:

$$\chi_S(D, w) \leq \rho_C(D, \hbar) \tag{6.77}$$

Let $\mathbb{z}$ be an optimal solution to the dual of the minimum cost survivable Steiner flow problem. From our assumption that $\chi_S(D, w) = 1$ along with $d = 1$, we have $\sum_{a \in A} \sum_{r \in R} \mathbb{z}_a^r = 1$. This implies that the couple of vectors $(w, \mathbb{z})$ is a feasible solution to the dual of the maximum survivable Steiner flow problem for the instance induced by capacity vector $\hbar$. Moreover, the value of this feasible solution is $\sum_{a \in A} \hbar_a w_a$ which is exactly $\chi_C(D, w)$ by definition of χ and $\hbar$. Therefore we have:

$$\rho_S(D, \hbar) \leq \chi_C(D, w) \tag{6.78}$$

Combining Equation (6.77) and Equation (6.78), we deduce that for any weight vector w, we can find a capacity vector $\hbar$ which satisfies:

$$g_\chi(D, w) \leq g_\rho(D, \hbar) \tag{6.79}$$

so that $\max_{w \geq 0} g_\chi(D, w) \leq \max_{c \geq 0} g_\rho(D, c)$, which concludes the proof. □

Observe that Theorem 31 is valid in both directed, bidirected, and undirected networks.

6.4.2.3 Bidirected and undirected networks

Combining Theorem 31 with Theorem 27 obtained in the previous chapter, we immediately deduce the following corollary:

Corollary 14. *Given an undirected network $G = (V, E)$ with source s, set of receivers R, positive capacity c_e for each edge e, the value of a maximum survivable coded flow is at most $2|E|$ times the one of a maximum survivable Steiner flow, namely $g_\rho \leq 2|E|$.*

Since the upper bound of value $2|E|$ on the survivable *cost* coding gain g_χ also stands for bidirected networks, so is the same upper bound on the value of the survivable coding gain g_ρ of a bidirected network.

6.4.2.4 Directed networks

We shall use the following lemma:

Lemma 16. *Given a* unit capacity *network $D = (V, A)$ with source s and set of receivers R, let ρ_C be the residual throughput of a maximum survivable coded flow, and let φ_C be the value of a maximum coded flow. Then $\rho_C = \varphi_C - 1$.*

Proof. Removing one arc from a minimum cut between the source and any receiver reduces the corresponding maximum flow value by one unit. Hence $\rho_C \leq \varphi_C - 1$. Let $\varkappa$ be a maximum coded flow obtained by sending a maximum flow from the source to each receiver. We have:

$$\rho(\varkappa) = \min_{(a,r)\in A\times R} \left\{ \sum_{p\in\mathcal{P}^r} \varkappa_p - \sum_{p\in\mathcal{P}^r_a} \varkappa_p \right\} \tag{6.80}$$

$$\geq \min_{r\in R} \sum_{p\in\mathcal{P}^r} \varkappa_p - \max_{(a,r)\in A\times R} \sum_{p\in\mathcal{P}^r_a} \varkappa_p \tag{6.81}$$

$$\geq \varphi_C - 1 \tag{6.82}$$

so that $\rho_C \geq \rho(\varkappa) \geq \varphi_C - 1$. □

As far as directed networks are concerned, combining Theorem 31 with Theorem 28 (obtained in the previous chapter) yields the following corollary:

Corollary 15. *For any real number $\eta \geq 1$, there exists an infinite family of directed networks such that the ratio between the value of a maximum survivable coded flow and the one of a maximum survivable Steiner flow satisfies $g_\rho \geq \eta$.*

We can deduce the following slightly stronger result by a direct proof:

Theorem 32. *For any real number $\eta \geq 1$, there exists an infinite family of directed networks with* unit capacities on each arc*, such that the ratio between the value of a maximum survivable coded flow and the one of a maximum survivable Steiner flow satisfies $g_\rho \geq \eta$.*

Proof. We prove this theorem by exhibiting a family of directed networks with unbounded survivable coding gain. We again use the family of directed combination networks which

was used in [83, 84] to show that the coding gain can be arbitrarily large. Recall that given two positive integers n and k with $k \in [n]$, the directed combination network $C(n,k)$ is made of three layers of vertices: the first layer with the source s, while the second layer is made of n intermediate nodes identified with $[n]$, whereas the third layer consists of a set R of $\binom{n}{k}$ receivers, indexed by the subsets of $[n]$ of size k. Those three layers are linked together by one arc from the source to each intermediate node, and one arc from an intermediate node to each receiver whose index (a subset of $[n]$) contains the index of this intermediate node (an element of $[n]$). Furthermore each arc a has a unit capacity, $c_a = 1$. The directed combination network $C(n,k)$, for $n = 4$ and $k = 3$, is depicted on Figure 6.5. From the proof of Theorem 8, the value of a maximum Steiner flow in network $C(n,k)$ is $\varphi_S = \frac{n}{n-k+1}$, while it is possible to design a maximum coded flow of value $\varphi_C = k$ in the same network. Since the network has unit capacities, $\rho_C = k-1$ by Lemma 16. We can build a Steiner flow $\mathbb{x}$ with residual throughput $\rho(\mathbb{x}) = \frac{(k-1)n}{k(n-k+1)}$, and nominal throughput $\frac{n}{n-k+1}$, by using nk Steiner trees, each tree conveying $\frac{1}{k(n-k+1)}$ units of flow, such that any arc of the network is used by at most $k(n-k+1)$ distinct trees, hence satisfying all capacity requirements. Such a Steiner flow, for the combination network $C(4,3)$, is represented on Figure 6.6. This implies $\rho_S \geq \frac{(k-1)n}{k(n-k+1)}$. We shall now prove that this residual throughput is best possible. Let $\mathbb{y}$ be the vector defined by setting for each arc a:

$$\mathbb{y}_a = \begin{cases} \dfrac{k-1}{k(n-k+1)} & \text{if } a \in \delta^+(s) \quad (6.83) \\ 0 & \text{otherwise} \quad (6.84) \end{cases}$$

We also define vector $\mathbb{z}$ by setting, for each arc a and each receiver r:

$$\mathbb{z}_a^r = \begin{cases} \dfrac{1}{k\binom{n}{k}} & \text{if } a \in \delta^-(r) \quad (6.85) \\ 0 & \text{otherwise} \quad (6.86) \end{cases}$$

We claim that the couple of vectors $(\mathbb{y}, \mathbb{z})$ is a feasible solution to the dual of the maximum survivable Steiner flow problem. To understand why, notice that for any receiver r, $|\delta^-(r)| = k$ so that:

$$\sum_{r \in R} \sum_{a \in A} \mathbb{z}_a^r = \sum_{r \in R} \sum_{a \in \delta^-(r)} \mathbb{z}_a^r = 1 \quad (6.87)$$

and Constraint (6.31) is satisfied. Recall from Lemma 3 that any Steiner tree t uses at

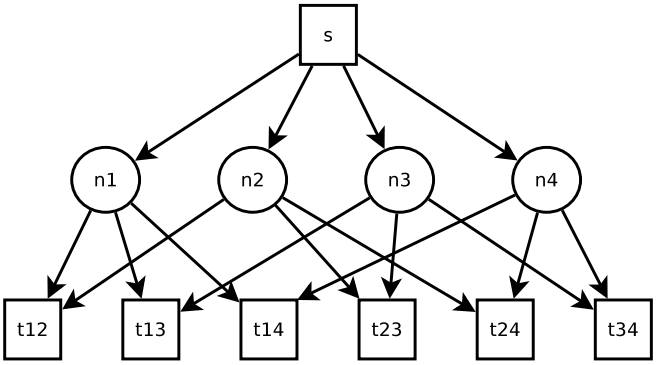


Figure 6.5 – The directed combination network $C(n, k)$, for $n = 4$ and $k = 3$.

least $n - k + 1$ intermediate nodes in order to span R, thus:

$$\sum_{a \in t} \mathbb{y}_a \geq \frac{k-1}{k} \tag{6.88}$$

Furthermore, for each Steiner tree t and each receiver r, there is exactly one arc in the set $p_t^r \cap \delta^-(r)$, which implies:

$$\sum_{r \in R} \sum_{a \in p_t^r} \mathbb{z}_a^r = \sum_{r \in R} \sum_{a \in p_t^r \cap \delta^-(r)} \mathbb{z}_a^r \tag{6.89}$$

$$= \sum_{r \in R} \frac{1}{k\binom{n}{k}} \tag{6.90}$$

$$= \frac{1}{k} \tag{6.91}$$

Combining Equation (6.88) and Equation (6.91), we obtain that Constraints (6.32) are satisfied. Since the network has unit capacities:

$$\sum_{a \in A} c_a \mathbb{y}_a = \frac{(k-1)n}{k(n-k+1)} \tag{6.92}$$

thus $\rho_S \leq \frac{(k-1)n}{k(n-k+1)}$ by weak duality. Hence $\rho_S = \frac{(k-1)n}{k(n-k+1)}$, which implies:

$$g_\rho = \frac{k(n-k+1)}{n} \tag{6.93}$$

By taking $n = 2k$, we obtain $g_\rho = \frac{k+1}{2}$. This last quantity is unbounded when the parameter k grows to infinity. Finally, given real number $\eta \geq 1$, the network $C(2k, k)$ with $k = \lceil 2\eta - 1 \rceil$ satisfies all requirements of the theorem. □

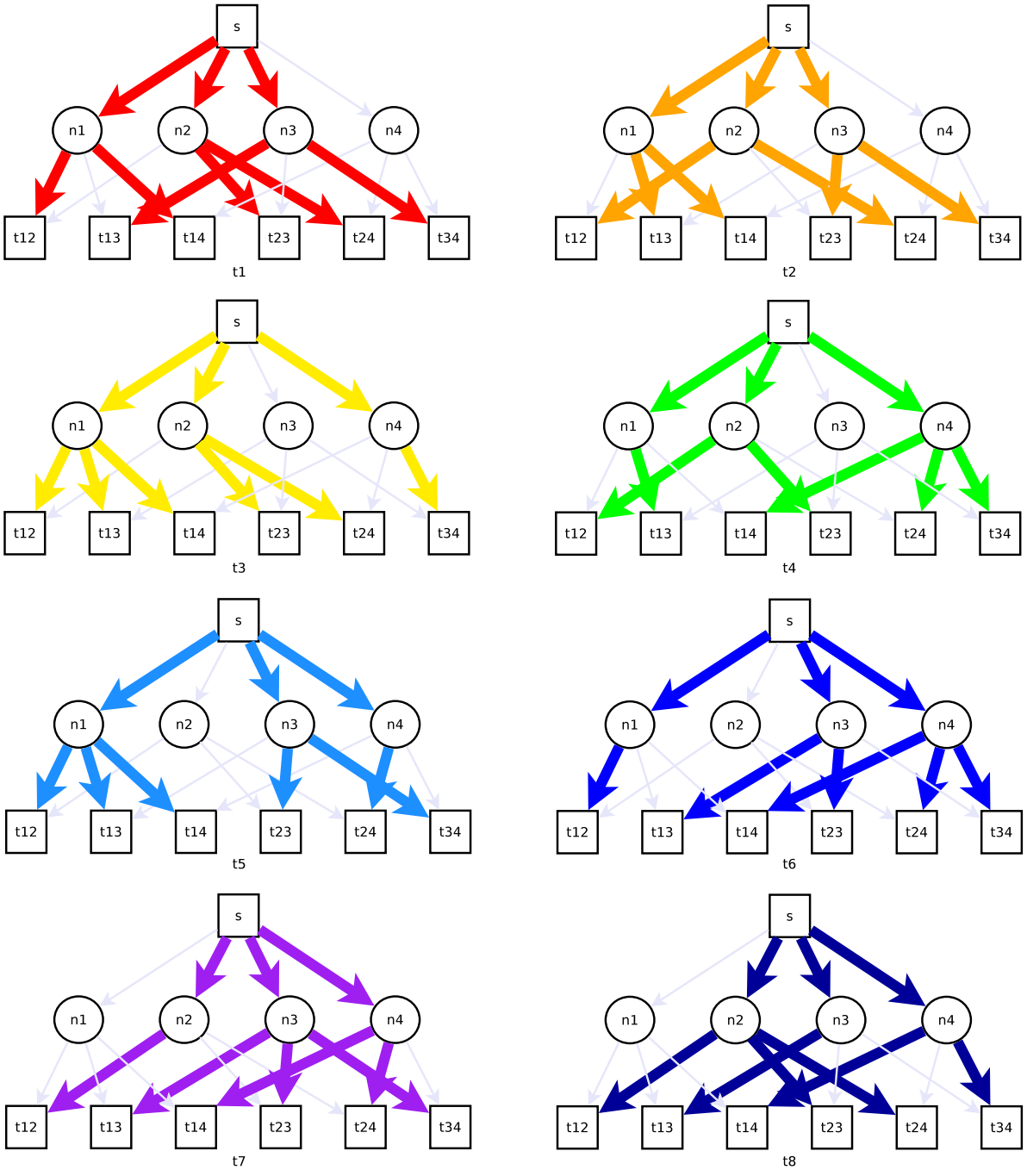


Figure 6.6 – A Steiner flow x made of eight supporting trees, each one routing $1/6$ units of flow, in the directed combination network $C(h, k)$, $h = 4$ and $k = 3$. This flow has a nominal throughput $\lambda(x) = 4/3$, and a residual throughput $\rho(x) = 2/3$.

The next theorem establishes a connection between the survivable coding gain g_ρ and the coding gain g_φ in a directed network with unit capacities.

Theorem 33. *In a directed network $D = (V, A)$ with* unit capacities*, if the coding gain g_φ equals* 1 *then the survivable coding gain g_ρ also equals* 1*.*

Proof. We denote by φ_S and φ_C respectively the value of a maximum Steiner flow and the one of a maximum coded flow, so that $g_\varphi = \frac{\varphi_C}{\varphi_S}$. Let $\mathbb{x}$ be a maximum Steiner flow. We have:

$$\rho(\mathbb{x}) = \min_{(a,r)\in A\times R} \left\{ \sum_{t\in\mathcal{T}} \mathbb{x}_t - \sum_{t\in\mathcal{T}_a^r} \mathbb{x}_t \right\} \tag{6.94}$$

$$\geq \sum_{t\in\mathcal{T}} \mathbb{x}_t - \max_{(a,r)\in A\times R} \sum_{t\in\mathcal{T}_a^r} \mathbb{x}_t \tag{6.95}$$

$$\geq \varphi_S - 1 \tag{6.96}$$

From Lemma 16 we have $\rho_C = \varphi_C - 1$. We deduce:

$$\varphi_S - 1 \leq \rho(\mathbb{x}) \leq \rho_S \leq \rho_C = \varphi_C - 1 \tag{6.97}$$

By assumption, $g_\varphi = 1$ so that $\varphi_S = \varphi_C$. Hence $\rho_S = \rho_C$ which concludes the proof. □

6.4.3 Experimental evaluation of the survivable coding gain

6.4.3.1 Setting

We use the same fifteen network topologies taken from the SNDlib library [97] as in the previous chapters. We also keep the source and the receivers as in the previous chapter. The capacity of each channel is picked in the set [10] (of all integers between 1 and 10) uniformly at random.

We compute each optimal Steiner flow by coupling a column generation algorithm with a subroutine solving the pricing problem thanks to the mixed-integer linear program 5.49 presented in the previous chapter. Each optimal coded flow problem is computed by solving its associated arc formulation thanks to a linear programming solver. All algorithms are implemented in Julia 0.3.8 [98, 99]. We use the Julia package *JuMP* [100] to call the open-source (mixed-integer) linear programming solver *CLP/CBC* [101].

6.4.3.2 Undirected networks

Some features of our fifteen instances are summarized in Table 6.2. For each instance, we recall the number of vertices $|V|$, the number of edges $|E|$, and the number of receivers $|R|$ in the network.

Instance	$\|V\|$	$\|E\|$	$\|R\|$
abilene	12	15	3
atlanta	15	22	5
france	25	45	6
geant	22	36	10
germany50	50	88	4
giul39	39	86	3
india35	35	80	4
newyork	16	49	4
nobel-eu	28	41	3
norway	27	51	5
pioro40	40	89	6
polska	12	18	5
ta1	24	55	6
ta2	65	108	4
zib54	54	81	4

Table 6.2 – Some features of the fifteen undirected instances.

Table 6.3 gives, for each undirected instance, information regarding the maximum survivable Steiner flow returned by the column generation algorithm. More precisely, column ρ_S provides the cost of the optimal Steiner flow. Column *Trees* gives the number of trees actually used by the optimal Steiner flow to convey some positive flow. Finally, column *Iterations* gives the number of iterations made by the column generation algorithm while column *Time* provides the associated running time expressed in seconds.

Table 6.4 is similar to Table 6.3 in that it provides, for each undirected instance, information regarding the maximum survivable coded flow returned by the linear programming solver. Column ρ_C gives the value of the optimal coded flow while column *Time* provides the associated running time expressed in seconds.

Notice that, for each undirected instance, the survivable coding gain g_ρ is equal to 1.

Instance	ρ_S	Trees	Iterations	Time (s)
abilene	1.0	3	5	0.76
atlanta	2.0	5	7	0.22
france	1.0	6	9	0.62
geant	2.0	5	11	0.93
germany50	7.0	22	32	2.55
giul39	6.0	13	24	1.18
india35	6.0	11	14	0.77
newyork	22.0	17	32	1.22
nobel-eu	8.0	8	11	0.27
norway	6.0	13	23	1.33
pioro40	7.0	18	36	3.94
polska	1.0	5	7	0.19
ta1	8.0	7	16	0.95
ta2	10.0	15	35	3.5
zib54	6.0	11	16	1.64

Table 6.3 – Features of the maximum survivable Steiner flow returned by the column generation algorithm for each undirected network.

Instance	ρ_C	Time (s)
abilene	1.0	0.01
atlanta	2.0	0.01
france	1.0	0.01
geant	2.0	0.02
germany50	7.0	0.02
giul39	6.0	0.07
india35	6.0	0.02
newyork	22.0	0.01
nobel-eu	8.0	0.01
norway	6.0	0.02
pioro40	7.0	0.04
polska	1.0	0.01
ta1	8.0	0.02
ta2	10.0	0.03
zib54	6.0	0.02

Table 6.4 – Features of the maximum survivable coded flow returned by the linear programming solver for each undirected network.

6.4.3.3 Bidirected networks

The test bench for bidirected instances is obtained by considering the bidirected network associated to each undirected instance presented in Table 6.2. All other features of the

undirected instances remain unchanged.

Table 6.5 gives, for each bidirected instance, information regarding the maximum survivable Steiner flow returned by the column generation algorithm. More precisely, column ρ_S provides the cost of the optimal Steiner flow. Column *Trees* gives the number of trees actually used by the optimal Steiner flow to convey some positive flow. Finally, column *Iterations* gives the number of iterations made by the column generation algorithm while column *Time* provides the associated running time expressed in seconds.

Instance	ρ_S	Trees	Iterations	Time (s)
abilene	1.0	3	5	0.28
atlanta	2.0	4	8	0.25
france	1.0	5	7	0.49
geant	2.0	6	15	1.77
germany50	7.0	16	27	2.25
giul39	6.0	7	23	1.33
india35	6.0	14	21	1.69
newyork	22.0	16	32	1.59
nobel-eu	8.0	6	11	0.34
norway	6.0	15	21	1.38
pioro40	7.0	20	35	4.99
polska	1.0	5	8	0.26
ta1	12.0	19	33	2.45
ta2	10.0	17	37	4.2
zib54	7.0	12	17	1.52

Table 6.5 – Features of the maximum survivable Steiner flow returned by the column generation algorithm for each bidirected network.

Table 6.6 provides, for each bidirected instance, information regarding the maximum survivable coded flow returned by the linear programming solver. Column ρ_C gives the value of the optimal coded flow while column *Time* provides the associated running time expressed in seconds.

For each bidirected instance, the survivable coding gain g_ρ is equal to 1.

6.4.3.4 Directed networks

Table 6.7 gives, for each directed instance, information regarding the maximum survivable Steiner flow returned by the column generation algorithm. More precisely, column

Instance	ρ_C	Time (s)
abilene	1.0	0.01
atlanta	2.0	0.01
france	1.0	0.02
geant	2.0	0.02
germany50	7.0	0.02
giul39	6.0	0.01
india35	6.0	0.01
newyork	22.0	0.01
nobel-eu	8.0	0.01
norway	6.0	0.02
pioro40	7.0	0.09
polska	1.0	0.0
ta1	12.0	0.02
ta2	10.0	0.02
zib54	7.0	0.02

Table 6.6 – Features of the maximum survivable coded flow returned by the linear programming solver for each bidirected network.

ρ_S provides the cost of the optimal Steiner flow. Column *Trees* gives the number of trees actually used by the optimal Steiner flow to convey some positive flow. Finally, column *Iterations* gives the number of iterations made by the column generation algorithm while column *Time* provides the associated running time expressed in seconds.

Table 6.8 provides, for each directed instance, information regarding the maximum survivable coded flow returned by the linear programming solver. Column ρ_C gives the value of the optimal coded flow while column *Time* provides the associated running time expressed in seconds.

Observe that, for each directed instance, the survivable coding gain g_ρ is equal to 1.

6.4.3.5 Analysis

For each type of information flow, and for each instance, the proposed algorithm finds an optimal solution in a matter of seconds. The benefit of using coding techniques to maximize the residual throughput seems to vanish regardless of the network structure. However, practical restrictions, like the number of multicast trees used to route the data, may favor the use of network coding over multicast alone, especially when dealing with

Instance	ρ_S	Trees	Iterations	Time (s)
abilene	1.0	3	5	0.09
atlanta	2.0	5	8	0.16
france	1.0	4	10	0.41
geant	2.0	4	12	0.83
germany50	7.0	16	30	1.65
giul39	6.0	10	23	0.87
india35	6.0	12	19	0.89
newyork	22.0	18	34	1.03
nobel-eu	8.0	7	10	0.2
norway	6.0	8	20	0.83
pioro40	7.0	21	34	3.04
polska	1.0	2	8	0.21
ta1	8.0	9	24	1.16
ta2	10.0	18	31	2.25
zib54	7.0	8	15	0.78

Table 6.7 – Features of the maximum survivable Steiner flow returned by the column generation algorithm for each directed network.

Instance	ρ_C	Time (s)
abilene	1.0	0.01
atlanta	2.0	0.01
france	1.0	0.01
geant	2.0	0.02
germany50	7.0	0.02
giul39	6.0	0.01
india35	6.0	0.01
newyork	22.0	0.01
nobel-eu	8.0	0.01
norway	6.0	0.01
pioro40	7.0	0.03
polska	1.0	0.0
ta1	8.0	0.07
ta2	10.0	0.02
zib54	7.0	0.02

Table 6.8 – Features of the maximum survivable coded flow returned by the linear programming solver for each directed network.

survivability issues where more trees are required in order to reduce the impact of failures.

6.5 Conclusion

In this chapter we consider networks prone to single link failures. We develop models to compute the maximum survivable throughput allowed by either a Steiner flow or a coded flow. We also provide algorithms to solve each problem. Furthermore, we define and study the survivable coding gain which makes comparisons between the two kinds of information flows. Finally, we derive both theoretical and numerical results on this survivable coding gain. The theoretical results are summarized in Table 6.9 below.

Structure	Coding gain	Survivable coding gain
Bidirected	$\{1\}$	$[1, 2\|L\|]$
Undirected	$[1, 2]$	$[1, 2\|E\|]$
Directed	$[1, +\infty[$	$[1, +\infty[$

Table 6.9 – Survivable coding gain domain for each network structure.

It could be interesting to generalize this study to the framework of multicommodity information flows.

Chapter 7

Conclusion

7.1 Wrap up

Throughout the present dissertation, we investigated models of information flows in telecommunication networks. The ability to perform basic operations on data, like duplication and coding, combined with the increasing computational power available at the nodes of a telecommunication network, open the way to new advanced routing techniques like *multicast* and *network coding*. Those promising techniques may help a telecommunication operator to cope with the emerging trends in the network usage. To this end, some new tools are required to help the practitioner in his decision process regarding the design and the features of its network. The aim of this thesis, is to propose a toolbox for the design and the optimization of some information flows in networks. By focusing our analysis on models and algorithms, we let all freedom to the telecommunication community to discuss and evaluate the benefit of deploying a given technique in a peculiar network. However, the tools proposed here may facilitate the design of efficient algorithms to solve some practical problems naturally arising in the framework of information flows.

Chapter 2 has been devoted to the presentation of the information flow framework used throughout the entire manuscript, including the very concept of *coding gain*. We also explained how some generic methods, like *column generation* or *network flow algorithms*, could be either adapted or successfully used as subroutines in the design of algorithms dealing with information flows.

In Chapter 3, we reviewed various results regarding *multicommodity* information flows. We explained wherein many results in this setting are simple extensions of the classical information flow framework. We also explained how the method originally proposed by Garg and Könemann [21], and later extended by Fleischer and Wayne [22], could be combined with an approximation algorithm to find minimum-cost Steiner trees as a subroutine, so as to solve some multicommodity Steiner flow problems with the same accuracy as the one provided by the subroutine. Finally, we defined and studied the *multicommodity coding gain* as a natural generalization of the aforementioned coding gain.

In Chapter 4 we tackled the issue of *congestion* in information flows, modelled as a *nonlinear* phenomenon. By exploiting some features of the studied problems, like *convexity*, we proposed practical algorithms, derived from the *conditional gradient method*, to find low cost information flows while satisfying a given demand. We also extend the definition of *cost coding gain* to the nonlinear setting while establishing a connection between the linear gain and the nonlinear one.

Chapter 5 and Chapter 6 are devoted to the study of information flows in the setting where any link of the network may fail. We propose models and algorithms, based on linear programming, to design *survivable* information flows satisfying some requirements with respect to the quality of service in the worst-case of failure. Although we are not able to derive the complexity of each Steiner flow problem studied in those two chapters, we can at least show that the associated pricing problem is NP-hard. We also define and study the *survivable cost coding gain* and the *survivable coding gain* respectively in Chapter 5 and Chapter 6. Finally, we exhibit a strong connection between those two gains, thanks to the theory of duality in linear programming.

Since the applications of multicast routing techniques in telecommunications often impose strong restrictions on the number of trees actually used to convey data in the network, we emphasize the use of iterative algorithms, like column generation or conditional gradient descent, which can produce a *sparse exact solution* in a reasonable amount of time, disregarding more sophisticated methods. As far as network coding is concerned, we constantly keep in mind the design of a coding scheme as a requirement to the practical implementation of coding techniques in a telecommunication network.

7.2 Future research

Now, we provide some research topics which we believe would be worth investigating.

We first give a word regarding the various coding gains defined in this thesis. To the best of our knowledge, it remains unclear whether the upper bound of 2 on the coding gain g_φ of an undirected network is actually tight. Thus, the question is also open for every upper bound derived from this one in the present document. Moreover, we may ask whether it is possible to get a multicommodity coding gain g_ϕ or g_λ independent of the number of commodity, a convex cost coding gain g_ψ which is not a function of the demand, or a constant survivable coding gain g_χ or g_ρ. Conversely, it would be interesting to exhibit another family of networks with provably high coding gains. Finally, we would like to know if it is possible to decide in polynomial time whether a coding gain differs from 1.

Speaking about tractability, the worst-case complexity of the minimum cost survivable Steiner flow problem and the one of the maximum survivable Steiner flow problem are still uncertain. It is also unclear whether it is possible to devise an approximation algorithm for one of those problems with a provable guarantee on the quality of the returned solution. The same question could be asked for the minimum convex cost Steiner flow problem. Finally, it would be interesting to design new algorithms for solving the various coded flow problems presented in this thesis. In this context, decentralized algorithms running in polynomial time would be particularly appealing.

At last, notice that in the present dissertation, we always assumed that every node of the network could perform all operations like duplication and coding. Since those operations require a specific device to be implemented at each node (at the very least a software), it is more likely that only a subset of the nodes will be endowed with the ability to perform those more complex operations, the others being limited to classical data forwarding. Many interesting problems naturally arise from this new setting.

Bibliography

[1] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001. 16, 47

[2] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Macmillan London, 1976. 16, 47

[3] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008. 16, 47, 98

[4] Gary Chartrand, Linda Lesniak, and Ping Zhang. *Graphs & digraphs*. CRC Press, 2010. 16, 47

[5] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001. 18, 49

[6] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012. 18, 49, 51

[7] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows. Technical report, DTIC Document, 1988. 18, 26, 49, 61, 62, 66, 92, 105, 106, 110, 112, 132, 159, 177, 201, 221, 251

[8] Beau Williamson. *Developing IP multicast networks*, volume 1. Cisco Press, 2000. 23, 54

[9] Katia Obraczka. Multicast transport protocols: a survey and taxonomy. *Communications Magazine, IEEE*, 36(1):94–102, 1998. 23, 54

[10] Laxman H Sahasrabuddhe and Biswanath Mukherjee. Multicast routing algorithms and protocols: a tutorial. *Network, IEEE*, 14(1):90–102, 2000. 23, 54

[11] Mahmood Hosseini, Dewan T Ahmed, Shervin Shirmohammadi, and Nicolas D Georganas. A survey of application-layer multicast protocols. *Communications Surveys & Tutorials, IEEE*, 9(3):58–74, 2007. 23, 54

[12] Carlos AS Oliveira and Panos M Pardalos. A survey of combinatorial optimization problems in multicast routing. *Computers & Operations Research*, 32(8):1953–1981, 2005. 23, 54

[13] Xiuzhen Cheng and Ding-Zhu Du. *Steiner trees in industry*, volume 11. Springer Science & Business Media, 2013. 23, 54

[14] Dingzhu Du and Xiaodong Hu. *Steiner tree problems in computer communication networks*. World Scientific, 2008. 23, 54

[15] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204–1216, 2000. 23, 26, 54, 56, 82, 92, 94, 132, 136

[16] Raymond W Yeung. *Information theory and network coding*. Springer Science & Business Media, 2008. 26, 57, 82

[17] C Fragouli and E Soljanin. Monograph on network coding: Fundamentals and applications. *Foundations and Trends in Networking*, 2(1), 2007. 26, 57

[18] Muriel Médard and Alex Sprintson. *Network coding: fundamentals and applications*. Academic Press, 2011. 26, 57

[19] Takahiro Matsuda, Taku Noguchi, and Tetsuya Takine. Survey of network coding and its applications. *IEICE transactions on communications*, 94(3):698–717, 2011. 26, 57

[20] Riccardo Bassoli, Hugo Marques, Jose Rodriguez, Kenneth W Shum, and Rahim Tafazolli. Network coding theory: A survey. *Communications Surveys & Tutorials, IEEE*, 15(4):1950–1978, 2013. 26, 57, 94

[21] Naveen Garg and Jochen Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007. 28, 118, 119, 123, 124, 270, 295, 305

[22] Lisa K Fleischer and Kevin D Wayne. Fast and simple approximation schemes for generalized flow. *Mathematical Programming*, 91(2):215–238, 2002. 28, 123, 270, 301, 305

[23] Luigi Fratta, Mario Gerla, and Leonard Kleinrock. The flow deviation method: An approach to store-and-forward communication network design. *Networks*, 3(2):97–133, 1973. 29

[24] April Rasala Lehman and Eric Lehman. Complexity classification of network information flow problems. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 142–150. Society for Industrial and Applied Mathematics, 2004. 57, 93

[25] Yuval Cassuto and Jehoshua Bruck. Network coding for non-uniform demands. In *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, pages 1720–1724. IEEE, 2005. 57, 125

[26] Randall Dougherty, Christopher Freiling, and Kenneth Zeger. Insufficiency of linear coding in network information flow. *Information Theory, IEEE Transactions on*, 51(8):2745–2759, 2005. 57

[27] Chandra Chekuri, Christina Fragouli, and Emina Soljanin. On achievable information rates in single-source non-uniform demand networks. In *Information Theory, 2006 IEEE International Symposium on*, pages 773–777. IEEE, 2006. 57, 125

[28] Michael Langberg and Alex Sprintson. On the hardness of approximating the network coding capacity. *Information Theory, IEEE Transactions on*, 57(2):1008–1014, 2011. 57

[29] Randall Dougherty and Kenneth Zeger. Nonreversibility and equivalent constructions of multiple-unicast networks. *Information Theory, IEEE Transactions on*, 52(11):5067–5077, 2006. 58

[30] TE Harris and FS Ross. Fundamentals of a method for evaluating rail net capacities. Technical report, DTIC Document, 1955. 62

[31] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956. 62

[32] Peter Elias, Amiel Feinstein, and Claude E Shannon. A note on the maximum flow through a network. *Information Theory, IRE Transactions on*, 2(4):117–119, 1956. 62

[33] James B Orlin. Max flows in o (nm) time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 765–774. ACM, 2013. 62

[34] Jonah Sherman. Nearly maximum flows in nearly linear time. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 263–269. IEEE, 2013. 62

[35] Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 217–226. SIAM, 2014. 62

[36] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in õ (vrank) iterations and faster algorithms for maximum flow. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 424–433. IEEE, 2014. 62

[37] Alexander Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445, 2002. 62

[38] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972. 66, 92

[39] Zongpeng Li, Baochun Li, Dan Jiang, and Lap Chi Lau. On achieving optimal throughput with network coding. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2184–2194. IEEE, 2005. 68, 69, 83, 88, 89, 90, 103, 129, 132, 134, 136

[40] Kamal Jain, Mohammad Mahdian, and Mohammad R Salavatipour. Packing steiner trees. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 266–274. Society for Industrial and Applied Mathematics, 2003. 69, 74, 78, 122, 123, 160

[41] Naveen Garg, Rohit Khandekar, Keshav Kunal, and Vinayaka Pandit. Bandwidth maximization in multicasting. In *Algorithms-ESA 2003*, pages 242–253. Springer, 2003. 69

[42] Xunrui Yin, Xin Wang, Jin Zhao, Xiangyang Xue, and Zongpeng Li. On benefits of network coding in bidirected networks and hyper-networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 325–333. IEEE, 2012. 76, 97, 98, 177, 186, 225

[43] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972. 77, 117, 120, 163

[44] Miroslav Chlebík and Janka Chlebíková. The steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science*, 406(3):207–214, 2008. 77

[45] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved lp-based approximation for steiner tree. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 583–592. ACM, 2010. 77

[46] Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 585–594. ACM, 2003. 77

[47] Moses Charikar, Chandra Chekuri, To-yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *Journal of Algorithms*, 33(1):73–91, 1999. 78

[48] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012. 78

[49] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956. 78, 171

[50] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389–1401, 1957. 78, 171

[51] R Kipp Martin. Using separation algorithms to generate mixed integer model reformulations. *Operations Research Letters*, 10(3):119–128, 1991. 78, 171

[52] Yoeng-Jin Chu and Tseng-Hong Liu. On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396, 1965. 78, 171

[53] Jack Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240, 1967. 78, 171

[54] Harold N Gabow and KS Manu. Packing algorithms for arborescences (and spanning trees) in capacitated graphs. *Mathematical Programming*, 82(1-2):83–109, 1998. 78

[55] Michel X Goemans and Young-Soo Myung. A catalog of steiner tree formulations. *Networks*, 23(1):19–28, 1993. 79, 120, 144, 164, 189, 215

[56] Yunnan Wu, Philip Chou, Kamal Jain, et al. A comparison of network coding and tree packing. In *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*, page 143. IEEE, 2004. 80, 81, 103

[57] Leonidas Georgiadis. Bottleneck multicast trees in linear time. *Communications Letters, IEEE*, 7(11):564–566, 2003. 81

[58] Shuo-Yen Robert Li, Raymond W Yeung, and Ning Cai. Linear network coding. *Information Theory, IEEE Transactions on*, 49(2):371–381, 2003. 83, 92

[59] Ralf Koetter and Muriel Médard. An algebraic approach to network coding. *Networking, IEEE/ACM Transactions on*, 11(5):782–795, 2003. 83, 93, 221, 222, 252, 253

[60] Zongpeng Li, Baochun Li, Dan Jiang, and Lap Chi Lau. On achieving optimal end-to-end throughput in data networks: Theoretical and empirical studies. *Univ. of Toronto, Dept. ECE, Tech. Rep*, 2004. 83, 88, 89, 90, 129, 132, 134, 136

[61] Zongpeng Li and Baochun Li. Network coding in undirected networks. CISS, 2004. 90, 97, 98, 99, 139

[62] Zongpeng Li and Baochun Li. Efficient and distributed computation of maximum multicast rates. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1618–1628. IEEE, 2005. 92

[63] Tracey Ho, David R Karger, Muriel Médard, and Ralf Koetter. Network coding from a network flow perspective. In *IEEE International Symposium on Information Theory*, pages 441–441, 2003. 93

[64] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *Information Theory, IEEE Transactions on*, 52(10):4413–4430, 2006. 93

[65] Sidharth Jaggi, Peter Sanders, Philip A Chou, Michelle Effros, Sebastian Egner, Kamal Jain, and Ludo MGM Tolhuizen. Polynomial time algorithms for multicast network code construction. *Information Theory, IEEE Transactions on*, 51(6):1973–1982, 2005. 93, 222

[66] Michael Langberg, Alexander Sprintson, and Jehoshua Bruck. Network coding: a computational perspective. *Information Theory, IEEE Transactions on*, 55(1):147–157, 2009. 93

[67] Minkyu Kim, Muriel Médard, Varun Aggarwal, Una-May O'Reilly, Wonsik Kim, Chang W Ahn, and Michelle Effros. Evolutionary approaches to minimizing network coding resources. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1991–1999. IEEE, 2007. 93

[68] Christina Fragouli and Emina Soljanin. Information flow decomposition for network coding. *Information Theory, IEEE Transactions on*, 52(3):829–848, 2006. 93

[69] Christina Fragouli and Emina Soljanin. A connection between network coding and convolutional codes. In *Communications, 2004 IEEE International Conference on*, volume 2, pages 661–666. IEEE, 2004. 93

[70] Sidharth Jaggi, Michelle Effros, T Ho, and Muriel Médard. On linear network coding. In *Proc. of the 42nd Allerton Conference*, 2004. 93

[71] Shuo-Yen Robert Li and Raymond W Yeung. On convolutional network coding. In *Information Theory, 2006 IEEE International Symposium on*, pages 1743–1747. IEEE, 2006. 93

[72] Elona Erez and Meir Feder. Efficient network code design for cyclic networks. *Information Theory, IEEE Transactions on*, 56(8):3862–3878, 2010. 93

[73] Michele Sanna and Ebroul Izquierdo. A survey of linear network coding and network error correction code constructions and algorithms. *International Journal of Digital Multimedia Broadcasting*, 2011, 2011. 94

[74] Jack Edmonds. Edge-disjoint branchings. *Combinatorial algorithms*, 9:91–96, 1973. 97

[75] H Bernhard, BH Korte, and J Vygen. Combinatorial optimization: Theory and algorithms, 2008. 97, 98

[76] Zongpeng Li, Baochun Li, and Lap Chi Lau. A constant bound on throughput improvement of multicast network coding in undirected networks. *Information Theory, IEEE Transactions on*, 55(3):1016–1026, 2009. 97, 98

[77] William Thomas Tutte. On the problem of decomposing a graph into n connected factors. *Journal of the London Mathematical Society*, 1(1):221–230, 1961. 98

[78] C St JA Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 1(1):445–450, 1961. 98

[79] Xunrui Yin, Yan Wang, Zongpeng Li, Xin Wang, Jin Zhao, and Xiangyang Xue. Bounding the advantage of multicast network coding in general network models. *Communications, IEEE Transactions on*, 62(3):1023–1032, 2014. 98, 186, 255

[80] András Frank. Edge-connection of graphs, digraphs, and hypergraphs. In *More sets, graphs and numbers*, pages 93–141. Springer, 2006. 99

[81] Amit Agarwal and Moses Charikar. On the advantage of network coding for improving network throughput. In *Information Theory Workshop, 2004. IEEE*, pages 247–249. IEEE, 2004. 99, 186

[82] Christina Fragouli and Emina Soljanin. Network coding fundamentals. *Foundations and Trends® in Networking*, 2(1):1–133, 2007. 99

[83] Peter Sanders, Sebastian Egner, and Ludo Tolhuizen. Polynomial time algorithms for network information flow. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 286–294. ACM, 2003. 99, 100, 187, 225, 259

[84] Chi Kin Ngai and Raymond W Yeung. Network coding gain of combination networks. In *Information Theory Workshop, 2004. IEEE*, pages 283–287. IEEE, 2004. 99, 100, 187, 225, 259

[85] Shreya Maheshwar, Zongpeng Li, and Baochun Li. Bounding the coding advantage of combination network coding in undirected networks. *Information Theory, IEEE Transactions on*, 58(2):570–584, 2012. 101, 187, 225

[86] Leonid Zosin and Samir Khuller. On directed steiner trees. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 59–63. Society for Industrial and Applied Mathematics, 2002. 102

[87] Chandra Chekuri, Christina Fragouli, and Emina Soljanin. On average throughput and alphabet size in network coding. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2410–2424, 2006. 102

[88] Eric Gourdin and Yuhui Wang. Some further investigation on maximum throughput: Does network coding really help? In *Teletraffic Congress (ITC 24), 2012 24th International*, pages 1–8. IEEE, 2012. 103

[89] Mohamed Saad, Tamás Terlaky, Anthony Vannelli, and Hu Zhang. Packing trees in communication networks. *Journal of combinatorial optimization*, 16(4):402–423, 2008. 117, 119

[90] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. 117, 210, 244

[91] Jochen Könemann et al. Fast combinatorial algorithms for packing and covering problems. 2000. 118

[92] José R Correa, Cristina G Fernandes, and Yoshiko Wakabayashi. Approximating a class of combinatorial problems with rational objective function. *Mathematical programming*, 124(1-2):255–269, 2010. 120

[93] Krzysztof Ciebiera, Piotr Godlewski, Piotr Sankowski, and Piotr Wygocki. Approximation algorithms for steiner tree problems based on universal solution frameworks. *arXiv preprint arXiv:1410.7534*, 2014. 120, 124

[94] Lisa K Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000. 123

[95] Andreas Baltz and Anand Srivastav. Fast approximation of minimum multicast congestion–implementation versus theory. *RAIRO-Operations Research-Recherche Opérationnelle*, 38(4):319–344, 2004. 124

[96] Klaus Jansen and Hu Zhang. Approximation algorithms for general packing problems and their application to the multicast congestion problem. *Mathematical Programming*, 114(1):183–206, 2008. 124, 155

[97] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. Sndlib 1.0—survivable network design library. *Networks*, 55(3):276–286, 2010. 143, 188, 228, 262

[98] Jeff Bezanson, Stefan Karpinski, Viral B Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. 2012. *arXiv preprint arXiv:1209.5145*, 292, 2012. 144, 189, 228, 262

[99] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computin. *arXiv preprint arXiv:1411.1607*, 2014. 144, 189, 228, 262

[100] Miles Lubin and Iain Dunning. Computing in operations research using julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015. 144, 189, 228, 262

[101] Jeffrey T Linderoth and Ted K Ralphs. Noncommercial software for mixed-integer linear programming. *Integer Programming: Theory and Practice*, 3:253–303, 2005. 144, 189, 229, 262

[102] Mohammad A Raayatpanah, H Salehi Fathabadi, Babak H Khalaj, and S Khodayifar. Minimum cost multiple multicast network coding with quantized rates. *Computer Networks*, 57(5):1113–1123, 2013. 154

[103] Santosh Vempala and Berthold Vöcking. Approximating multicast congestion. In *Algorithms and Computation*, pages 367–372. Springer, 1999. 155

[104] Qiang Lu and Hu Zhang. Implementation of approximation algorithms for the multicast congestion problem. In *Experimental and Efficient Algorithms*, pages 152–164. Springer, 2005. 155

[105] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 159, 177

[106] Adamou Ouorou, Philippe Mahey, and J-Ph Vial. A survey of algorithms for convex multicommodity flow problems. *Management Science*, 46(1):126–147, 2000. 159, 167, 173, 182

[107] Dimitri P Bertsekas, Robert G Gallager, and Pierre Humblet. *Data networks*, volume 2. Prentice-Hall International New Jersey, 1992. 159

[108] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956. 167

[109] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 427–435, 2013. 167, 170, 181

[110] Masao Fukushima. A modified frank-wolfe algorithm for solving the traffic assignment problem. *Transportation Research Part B: Methodological*, 18(2):169–177, 1984. 167

[111] Dimitri P Bertsekas. Nonlinear programming. 1999. 167, 173, 182

[112] Mokhtar S Bazaraa, Hanif D Sherali, and Chitharanjan Marakada Shetty. *Nonlinear programming: theory and algorithms*. John Wiley & Sons, 2013. 169, 180, 189

[113] Jo Bo Rosen. The gradient projection method for nonlinear programming. part i. linear constraints. *Journal of the Society for Industrial & Applied Mathematics*, 8(1):181–217, 1960. 173, 182

[114] Dimitri P Bertsekas, Bob Gendron, and Wei Kang Tsai. Implementation of an optimal multicommodity network flow algorithm based on gradient projection and a path flow formulation. Technical report, DTIC Document, 1984. 173, 182

[115] Dimitri P Bertsekas, Robert G Gallager, and Pierre Humblet. *Data networks*, volume 2. Prentice-hall Englewood Cliffs, NJ, 1987. 173, 182

[116] Desmond S Lun, Niranjan Ratnakar, Muriel Médard, Ralf Koetter, David R Karger, Tracey Ho, Ebad Ahmed, and Fang Zhao. Minimum-cost multicast over coded packet networks. *Information Theory, IEEE Transactions on*, 52(6):2608–2623, 2006. 176, 177, 197

[117] Sandeep Bhadra, Sanjay Shakkottai, and Piyush Gupta. Min-cost selfish multicast with network coding. *Information Theory, IEEE Transactions on*, 52(11):5077–5087, 2006. 177

[118] Desmond S Lun, Niranjan Ratnakar, Ralf Koetter, Muriel Médard, Erfan Ahmed, and Hyunjoo Lee. Achieving minimum-cost multicast: A decentralized approach based on network coding. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1607–1617. IEEE, 2005. 197

[119] Sebastian Orlowski and Michal Pióro. On the complexity of column generation in survivable network design with path-based survivability mechanisms. *Zuse Institute Berlin and Warsaw University of Technology, Tech. Rep*, 2008. 201

[120] Sebastian Orlowski and Michał Pióro. Complexity of column generation in network design with path-based survivability mechanisms. *Networks*, 59(1):132–147, 2012. 201

[121] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980. 206

[122] Craig A Tovey. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984. 208

[123] Dritan Nace, Michał Pióro, Artur Tomaszewski, and Mateusz Żotkiewicz. Complexity of a classical flow restoration problem. *Networks*, 62(2):149–160, 2013. 210, 244

[124] Yigal Bejerano, Suman Jana, and Pramod V Koppol. Efficient construction of directed redundant steiner trees. In *Local Computer Networks (LCN), 2012 IEEE 37th Conference on*, pages 119–127. IEEE, 2012. 212

[125] Muriel Médard, Steven G Finn, and Richard A Barry. Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs. *IEEE/ACM Transactions on Networking (TON)*, 7(5):641–652, 1999. 212

[126] Yigal Bejerano and Pramod V Koppol. Link-coloring based scheme for multicast and unicast protection. In *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*, pages 21–28. IEEE, 2013. 212

[127] Yigal Bejerano and Pramod V Koppol. Optimal construction of redundant multicast trees in directed graphs. In *INFOCOM 2009, IEEE*, pages 2696–2700. IEEE, 2009. 213

[128] Fang Li and Wangmei Guo. An efficient polynomial time algorithm for robust multicast network code construction. *Communications Letters, IEEE*, 19(2):143–146, 2015. 222

[129] Yash P Aneja, R Chandrasekaran, and KPK Nair. Maximizing residual flow under an arc destruction. *Networks*, 38(4):194–198, 2001. 238, 239, 252

[130] Dimitris Bertsimas, Ebrahim Nasrabadi, and Sebastian Stiller. Robust and adaptive network flows. *Operations Research*, 61(5):1218–1242, 2013. 239

[131] Dimitris Bertsimas, Ebrahim Nasrabadi, and James B Orlin. On the power of randomization in network interdiction. *arXiv preprint arXiv:1312.3478*, 2013. 239

[132] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002. 290

[133] Jacques Desrosiers and Marco E Lübbecke. *A primer in column generation*. Springer, 2005. 291

[134] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998. 292

[135] Serge A Plotkin, David B Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995. 305

[136] Maurice Sion et al. On general minimax theorems. *Pacific J. Math*, 8(1):171–176, 1958. 305

Appendix A

Basics of linear programming

A.1 Linear programming

A.1.1 A brief history

We first quickly review the history of *linear programming*. This field, also referred to as *linear optimization* in the literature, is concerned with the *maximization* (or the *minimization*) of a linear mapping over the sub-set of a vector space induced by a set of linear inequalities. It takes its roots around 1940, in the pioneering works of Kantorovich, Koopmans and Hitchcock on some economic-related problems. Linear optimization was later extended by Dantzig, who proposed, in 1947, the so-called *simplex algorithm* as a generic method to solve the main problem of this field. In 1979, Khachiyan introduced the *ellipsoid algorithm* as the first method to solve linear programming problems in polynomial time with respect to the instance size. A major breakthrough in the field came from the *interior point method* devised by Karmarkar in 1984. Linear programming currently remains a very active research field while enjoying a wide use in industrial and economic applications.

A.1.2 Problem statement

We shall focus on the case where the linear mapping is to be maximized, without loss of generality, since switching between maximization and minimization of a function ϕ over a set X is a matter of replacing said function by its opposite, $\min_{x \in X} \phi(x) = -\max_{x \in X} -\phi(x)$. An instance of a linear optimization problem is made of a vector $c \in \mathbb{R}^n$,

a vector $b \in \mathbb{R}^m$ and a matrix $A \in \mathbb{R}^{m \times n}$ where m and n are two positive integers. The task at hand is to find a non-negative vector $x \in \mathbb{R}^n$ satisfying the inequalities $Ax \leq b$ and such that the quantity $c^T x$ is as large as possible (where ^{T} denotes the transpose operator). The vector x can be regarded as a collection of *variables* to be adjusted so as to maximize the value $c^T x$, it is often referred to as the *variable vector*. A non-negative vector x is *feasible*, or a *feasible solution*, if it satisfies each linear inequality, also referred to as a *constraint*, induced by the overall inequalities $Ax \leq b$. Let $\mathcal{X}$ be the set of all feasible vectors, for this peculiar instance of the optimization problem. For the sake of simplicity, we shall assume in what follows that this set is non-empty so that the optimization problem has at least one feasible solution with respect to the considered instance. It is possible to formalize a generic linear maximization problem as follows:

Problem	*Generic linear programming (canonical form)*
Instance	Objective vector $c \in \mathbb{R}^n$, constraints vector $b \in \mathbb{R}^m$, and constraints matrix $A \in \mathbb{R}^{m \times n}$
Solution	A feasible vector $x \in \mathcal{X} = \{x \in \mathbb{R}^n : Ax \leq b\}$
Objective	Maximize the value $c^T x$ of vector x

In the following, we denote by z^* the value of a maximum feasible vector x^* for the considered instance. Such a vector x^* is an *optimal* solution to (this instance of) the problem. The overall optimization problem can be summarized thanks to the following notation, referred to as the *linear program* associated to this formulation of the problem:

$$\begin{cases} z^* = \max \quad c^T x & \text{(A.1)} \\ \text{s.t.} \quad Ax \leq b & \text{(A.2)} \\ \qquad x \geq 0 & \text{(A.3)} \end{cases}$$

A.1.3 Duality

We shall refer to the previous maximization problem as the *primal problem*. To this linear optimization problem can be associated another one, called the *dual problem*, and

whose linear program is:

$$\begin{cases} d^* = \min \quad b^T y & \text{(A.4)} \\ \text{s.t.} \quad A^T y \geq c & \text{(A.5)} \\ \qquad y \geq 0 & \text{(A.6)} \end{cases}$$

Here, the non-negative vector $y \in \mathbb{R}^m$ is the collection of all variables, where the objective is to minimize the quantity $b^T y$ over the set defined by the inequality $A^T y \geq c$. In the following, we shall further assume that the dual problem is feasible. Let x, respectively y, be a feasible solution to the primal, respectively dual, problem. The *weak duality* theorem states that the corresponding objective value of the primal problem is not greater than the one of the dual problem:

$$c^T x \leq (A^T y)^T x = (Ax)^T y \leq b^T y \tag{A.7}$$

where the first, respectively second, inequality comes from the feasibility of y, respectively x, with respect to the dual, respectively primal, problem combined with the non-negativity of x, respectively y. The *strong duality* theorem further states that, for a couple (x^*, y^*) of *optimal* primal/dual vectors, the primal solution value actually equals the dual solution value, namely $c^T x^* = b^T y^*$. Finally, observe that the dual problem can also be modelled by the following linear program:

$$\begin{cases} d^* = -\max \quad -b^T y & \text{(A.8)} \\ \text{s.t.} \quad (-A)^T y \leq -c & \text{(A.9)} \\ \qquad y \geq 0 & \text{(A.10)} \end{cases}$$

Applying the previous dual transformation to this last linear program yields the subsequent one:

$$\begin{cases} z^* = -\min \quad -c^T x & \text{(A.11)} \\ \text{s.t.} \quad -Ax \geq -b & \text{(A.12)} \\ \qquad x \geq 0 & \text{(A.13)} \end{cases}$$

which is nothing but a reformulation of the primal problem. Hence, the dual problem of the dual problem is the primal problem. Given a pair of primal/dual problems, each of those two roles is rather a matter of convention.

A.1.4 Integer linear programming

We may also have to deal with *integer linear programming*. A problem of this class is obtained from a linear optimization problem by imposing that each component of a candidate solution x is integer:

$$\begin{cases} z^* = \max \quad c^T x & \text{(A.14)} \\ \text{s.t.} \quad Ax \leq b & \text{(A.15)} \\ \qquad x \in \mathbb{Z}_+^n & \text{(A.16)} \end{cases}$$

The linear program obtained by replacing the set of integrality constraints $x \in \mathbb{Z}_+^n$ by $x \in \mathbb{R}_+^n$ is called the *continuous relaxation* of the integer programming problem. If only a subset among all components of vector x are restricted to lie in $\mathbb{Z}_+$, one gets a *mixed integer linear programming* problem. Since those integrality requirements actually break the structure of the generic linear program (the set of all feasible solutions is no longer convex), it should not be surprising to find that a (mixed) integer linear programming problem is usually considerably harder to solve than its continuous counterpart. Actually, given a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$, the problem of deciding whether there exists a binary vector x satisfying the inequality $Ax \leq b$ is already NP-hard in the strong sense [132]. Although this rules out the possibility to find an algorithm running in polynomial time to solve any generic integer linear programming problem (unless P = NP), practically efficient methods to solve, or at least to approximate, this class of problems have been devised in the last decades. A very popular one is the so-called *branch and bound* approach, which often rests on the ability to solve efficiently the continuous relaxation of the integer programming problem.

A.1.5 Column generation

We shall now briefly describe this method with an emphasis on its application to our problem. Consider the following generic linear program:

$$\begin{cases} \max \; c^T x & \text{(A.17)} \\ \text{s.t.} \quad Ax \leq b & \text{(A.18)} \\ \qquad x \geq 0 & \text{(A.19)} \end{cases}$$

where we assume that the number of components of vector x is too huge to cope with explicitly, while the number of linear inequalities induced by Constraints (A.18) remains tractable (read polynomial in the instance size). The dual of this linear program is:

$$\begin{cases} \min\ b^T y & \text{(A.20)} \\ \text{s.t.} \quad A^T y \geq c & \text{(A.21)} \\ \qquad y \geq 0 & \text{(A.22)} \end{cases}$$

Given vector y, the separation problem associated to the dual program, also referred to as the *pricing problem*, is to decide whether Constraints (A.21) are all satisfied by vector y. Defining the vector of *reduced costs* as $c - A^T y$, the pricing problem amounts to look for a positive component of this last vector. Observe that, by duality, there is one such constraint for each component of vector x, hence there is a gigantic number of dual constraints to check.

We shall now describe the column generation algorithm. The main idea is to work with a restricted subset of all primal variables (the components of vector x). Initialization is performed by looking for a feasible primal solution x. We implicitly assume that this solution uses only a small subset of *active* variables. Let y be the corresponding dual vector. The next step is to solve the pricing problem with vector y. A violated constraint in the dual induces a variable, or *column* (of matrix A), to be added to the set of active variables. If such a violated constraint is found, update the set of active variables accordingly and solve the new primal problem thus obtained. Repeat the previous procedure until the current dual vector is feasible with respect to Constraints (A.21), which means that the vector of reduced costs is non-positive. In this last case, we get a feasible solution x to the primal problem, and a feasible solution y to the dual problem, satisfying the complementary slackness conditions. By linear programming duality, x is an optimal solution to the primal problem so the procedure stops and returns x. Observe that although this method eventually returns a feasible solution, assuming the primal problem is feasible, it may involve an exponential number of iterations. The reader interested in a more detailed presentation of the column generation method and its applications is referred to consult any book on this topic, [133] for example.

A.1.6 Summary

Linear programming is our core tool to model and compare the various optimization problems encountered along the manuscript. In some situations, linear programming is actually our only way to properly define a problem while avoiding intricate troubles. Furthermore, formulating an optimization problem as a linear program immediately provides powerful tools, like the aforementioned theory of duality, along with efficient and well-understood algorithms and methods. We refer the interested reader to the various books dealing with linear programming, like Schrijver's comprehensive treatise of the topic [134].

A.2 Linear programming and the maximum flow problem

We prove that the arc formulation and the path formulation of the maximum flow problem are equivalent in that both yield the same optimal value. Given a flow x under path formulation, define a flow f under arc formulation by setting:

$$f_a = \sum_{p \in \mathcal{P}_a} x_p \tag{A.23}$$

We first prove that f is a feasible flow with respect to the arc formulation. It should be clear that vector f satisfies the capacity requirements assuming that vector x does. Furthermore, for any vertex v different from the source s or the receiver r:

$$\begin{aligned}
\sum_{a \in \delta^-(v)} f_a &= \sum_{a \in \delta^-(v)} \sum_{p \in \mathcal{P}_a} x_p && \text{(A.24)} \\
&= \sum_{p \in \mathcal{P}} \sum_{a \in \delta^-(v) \cap p} x_p && \text{(A.25)} \\
&= \sum_{p \in \mathcal{P}} \sum_{a \in \delta^+(v) \cap p} x_p && \text{(A.26)} \\
&= \sum_{a \in \delta^+(v)} \sum_{p \in \mathcal{P}_a} x_p && \text{(A.27)} \\
&= \sum_{a \in \delta^+(v)} f_a && \text{(A.28)}
\end{aligned}$$

with the convention that a sum over the empty set equals 0. The first and the last equalities come from Equation (A.23), while the third equality results from the simple fact that for any path p the sets $\delta^-(v) \cap p$ and $\delta^+(v) \cap p$ are either simultaneously empty or singletons

(recall that v is distinct from s and r). Thus, vector f also satisfies all conservation constraints. Moreover we have:

$$
\begin{aligned}
|f| &= \sum_{a \in \delta^+(s)} f_a - \sum_{a \in \delta^-(s)} f_a && \text{(A.29)} \\
&= \sum_{a \in \delta^+(s)} \sum_{p \in \mathcal{P}_a} x_p - \sum_{a \in \delta^-(s)} \sum_{p \in \mathcal{P}_a} x_p && \text{(A.30)} \\
&= \sum_{p \in \mathcal{P}} \Big(\sum_{a \in \delta^+(s) \cap p} x_p - \sum_{a \in \delta^-(s) \cap p} x_p \Big) && \text{(A.31)} \\
&= \sum_{p \in \mathcal{P}} x_p && \text{(A.32)} \\
&= |x| && \text{(A.33)}
\end{aligned}
$$

where the second equality comes from Equation (A.23) and the fourth one can be deduced by observing that any path p uses exactly one arc to leave the source and never comes back to it since p is loop-free. Hence, the value of vector f equals the one of vector x. Combining all previous results, it is always possible to define a feasible solution f to the arc formulation from a feasible solution x to the path formulation, such that the value of f equals the one of x. Therefore, the value φ_F of a maximum flow with respect to the arc formulation is greater than, or equal to, the value $\varphi_{\mathcal{P}}$ of a maximum flow with respect to the path formulation. We will use the theory of duality in linear programming to justify the other inequality. We first provide the dual of each formulation of the maximum flow problem, starting with the arc formulation. Let y be the dual vector associated to Constraints (2.7), and let z be the one associated to Constraints (2.8). The dual of the linear program associated to the arc formulation is:

$$
\begin{cases}
\varphi_F = \min \displaystyle\sum_{(u,v) \in A} c_{(u,v)} y_{(u,v)} & \text{(A.34)} \\
\text{s.t.} \quad z_r - z_s \geq 1 & \text{(A.35)} \\
\qquad y_{(u,v)} + z_u - z_v \geq 0 \quad \forall (u,v) \in A & \text{(A.36)} \\
\qquad y_{(u,v)} \geq 0 \quad \forall (u,v) \in A & \text{(A.37)} \\
\qquad z_v \in \mathbb{R} \quad \forall v \in V & \text{(A.38)}
\end{cases}
$$

We also consider the dual of the linear program associated to the path formulation:

$$\begin{cases} \varphi_{\mathcal{P}} = \min \quad \sum_{a \in A} c_a y_a & \text{(A.39)} \\ \text{s.t.} \quad \sum_{a \in p} y_a \geq 1 \quad \forall p \in \mathcal{P} & \text{(A.40)} \\ \qquad y_a \geq 0 \quad \forall a \in A & \text{(A.41)} \end{cases}$$

where y is the vector of dual variables associated to Constraints (2.18). Consider an optimal solution y to the dual of the linear program associated to the path formulation. By strong duality applied to the primal / dual path formulation linear programs, the value of y is exactly the one of a maximum flow with respect to the path formulation, namely $\varphi_{\mathcal{P}}$. To define a feasible solution to the dual of the linear program associated to the arc formulation, just keep the vector y and, for any vertex v, set the value of variable z_v to the length of a shortest path from the source s to v, with length y_a on each arc a. By definition of a shortest path one has $z_v \leq z_u + y_{(u,v)}$ for any arc (u, v) and thus Constraints (A.36) are satisfied. From Constraints (A.40), the value of a shortest path from the source s to the receiver r is at least 1. This implies $z_r \geq 1$. Also notice that $z_s = 0$. Hence, Constraint (A.35) is satisfied. Finally (y, z) is a feasible solution to the dual of the linear program associated to the arc formulation with the same value as y, namely $\varphi_{\mathcal{P}}$. This implies that $\varphi_F \leq \varphi_{\mathcal{P}}$ by weak duality applied to the primal / dual arc formulation linear programs. Finally $\varphi_F = \varphi_{\mathcal{P}}$.

Appendix B

Approximation algorithms

B.1 The framework of Garg and Könemann

The presentation below is directly inspired from [21].

B.1.1 Reformulation of the covering problem

We will use the following reformulation of the covering problem. Given covering variable vector y, we denote by $D(y) = b^T y$ its corresponding value. Let β be the optimal value of the following problem:

$$\beta = \min_{y \geq 0} \left\{ \frac{D(y)}{\sigma(y)} \right\} \tag{B.1}$$

The next lemma shows that this last problem is actually a reformulation of the covering problem.

Lemma 17. *Given an optimal solution y^* to the covering problem, we have $\beta = D(y^*)$.*

Proof. Let y^* be an optimal solution to the covering problem. Since $\sigma(y^*) \geq 1$ by feasibility of y^* we get $D(y^*) \geq \frac{D(y^*)}{\sigma(y^*)} \geq \beta$. Now, let y be a non-negative vector such that $\beta = \frac{D(\mathrm{y})}{\sigma(\mathrm{y})}$. From our previous assumption on the feasibility of the covering problem, the quantity β is finite, hence $\sigma(\mathrm{y}) > 0$. Without loss of generality we can assume $\sigma(\mathrm{y}) = 1$ since otherwise we could just divide y by $\sigma(\mathrm{y})$ to obtain a new solution where this assumption is satisfied. Hence, y is a feasible solution to the covering problem and we get $\beta = D(\mathrm{y}) \geq D(y^*)$. □

B.1.2 Framework of the algorithm

Assume we have an approximation algorithm which, given a dual vector y, returns a column q such that $A_q^T y/c_q \leq \alpha\sigma(y)$. We now describe how to alter the method originally proposed by Garg and Könemann to get an approximation algorithm for the packing problem, using the aforementioned approximation algorithm as a subroutine.

Let δ and ϵ be two positive real numbers whose values will be suitably chosen later. The algorithm proceeds in iterations which we index by l. Start from the primal solution $x^{(0)} = 0$ and the potentially infeasible dual solution $y_i^{(0)} = \delta/b_i$ for all i. In iteration l, the algorithm first looks for a provably good column q of A by making a call to the oracle subroutine with input $y^{(l-1)}$. Observe that, even if the dual constraint corresponding to column q is not violated, there is no guarantee that all dual constraints are satisfied, unless $\alpha = 1$. The algorithm then determines a row p of A such that $b_p/A(p,q) = \min_{i\in I}\{b_i/A(i,q)\}$ and increases x_q by $b_p/A(p,q)$. Hence, the algorithm increases the value of variable x_q as much as possible without violating any constraint in the *original* packing problem. Observe that this increase in iteration l is independent from the choices made by the algorithm at previous iterations, which implies that the primal vector x is likely to be infeasible with respect to the packing constraints. Let $z^{(l)}$ be the *value* of the primal vector constructed by the algorithm at the end of iteration l. Thus $z^{(l)} = z^{(l-1)} + c_q b_p/A(p,q)$ holds with $z^{(0)} = 0$. The dual solution y is updated by setting for all i

$$y_i^{(l)} = y_i^{(l-1)} \left(1 + \epsilon \frac{b_p/A(p,q)}{b_i/A(i,q)}\right) \tag{B.2}$$

For brevity, we use the notation $D(l) = D(y^{(l)})$ and $\sigma(l) = \sigma(y^{(l)})$ for each iteration l. The procedure terminates after τ iterations, where τ is the smallest integer such that $D(\tau) \geq 1$. Observe once again that the resulting primal vector is likely to be infeasible with respect to the packing constraints. In order to get a feasible solution to the packing problem, we will scale the vector returned by the algorithm by a suitable value, as emphasized by the next lemma.

Lemma 18. *The vector obtained by scaling down by a factor of* $\log_{1+\epsilon}(\frac{1+\epsilon}{\delta})$ *the vector* x *returned by the previous algorithm is a feasible solution to the packing problem.*

Proof. We show that the scaled vector satisfies the packing constraints. When, at iteration l, the algorithm picks column q and increases variable x_q by the amount $b_p/A(p,q)$, the left-hand-side of the i^{th} primal constraint is increased by $b_p A(i,q)/A(p,q)$ for each i. By definition of p this last quantity is upper-bounded by b_i, or equivalently the quantity $\nu_i^{(l)} = b_p A(i,q)/b_i A(p,q)$ satisfies $\nu_i^{(l)} \leq 1$. Simultaneously, the dual variable y_i is multiplied by a factor $1+\epsilon\nu_i$. Since for any ν such that $0 \leq \nu \leq 1$, $1+\epsilon\nu \geq (1+\epsilon)^\nu$ (using the Taylor series of $(1+\epsilon)^\nu$), we get:

$$\prod_{l=1}^{\tau}(1+\epsilon\nu_i^{(l)}) \geq (1+\epsilon)^{\sum_{l=1}^{\tau}\nu_i^{(l)}} \tag{B.3}$$

At iteration l, the i^{th} dual variable increases by a factor $1+\epsilon\nu_i^{(l)}$ which is smaller than $1+\epsilon$ since $\nu_i^{(l)} \leq 1$. Initially $y_i^{(0)} = \delta/b_i$. After $\tau-1$ iterations, we have by definition of τ, $b_i y_i^{(\tau-1)} \leq D(\tau-1) < 1$ (since $b_i \geq 0$ for each i), hence $y_i^{(\tau-1)} < 1/b_i$ which implies $y_i^{(\tau)} < (1+\epsilon)/b_i$. The value of the left-hand-side of the i^{th} packing constraint at the end of the algorithm is equal to $\sum_{l=1}^{\tau}\nu_i^{(l)} b_i$. Moreover,

$$y_i^{(\tau)} = y_i^{(0)} \prod_{l=1}^{\tau}(1+\epsilon\nu_i^{(l)}) \tag{B.4}$$

hence, from the definition of $y_i^{(0)}$:

$$\prod_{l=1}^{\tau}(1+\epsilon\nu_i^{(l)}) = \frac{y_i^{(\tau)}}{y_i^{(0)}} < \frac{1+\epsilon}{\delta} \tag{B.5}$$

which, combined with Equation (B.3), directly implies that:

$$\sum_{l=1}^{\tau}\nu_i^{(l)} < \log_{1+\epsilon}(\frac{1+\epsilon}{\delta}) \tag{B.6}$$

Therefore, the left-hand-side of the i^{th} packing constraint is at most $b_i \log_{1+\epsilon}((1+\epsilon)/\delta)$, and scaling down the vector returned by the algorithm by $\log_{1+\epsilon}((1+\epsilon)/\delta)$ gives a feasible solution to the packing problem. □

B.1.3 Approximation guarantee

The next lemma states that the feasible solution to the packing problem obtained after scaling has almost the same guarantee on its quality than the guarantee provided by the oracle used as a subroutine.

Lemma 19. *Given an oracle with ratio α to solve the separation problem (associated to the covering problem), the algorithm can be tuned to return a feasible solution to the packing problem whose value is no less than $(1-\omega)/\alpha$ times the optimal one, where ω is any fixed positive real number.*

Proof. Let α be the quality ratio guaranteed by the subroutine. We first show that $\beta/z^{(\tau)} \leq \alpha\epsilon/\ln((m\delta)^{-1})$. At iteration $l \geq 1$, we have:

$$D(l) = \sum_{i\in I} b_i y_i^{(l)} \tag{B.7}$$

$$= \sum_{i\in I} b_i y_i^{(l-1)} + \epsilon \frac{b_p}{A(p,q)} \sum_{i\in I} A(i,q) y_i^{(l-1)} \tag{B.8}$$

$$= D(l-1) + \epsilon(z^{(l)} - z^{(l-1)}) A_q^T y^{(l-1)}/c_q \tag{B.9}$$

$$\leq D(l-1) + \alpha\epsilon(z^{(l)} - z^{(l-1)})\sigma(l-1) \tag{B.10}$$

where the inequality comes from the approximation guarantee of the subroutine. By induction on l, we have:

$$D(l) \leq D(0) + \alpha\epsilon \sum_{h=1}^{l} (z^{(h)} - z^{(h-1)})\sigma(h-1) \tag{B.11}$$

Since $D(0) = m\delta$ and, by definition of β, for each iteration l, $\beta \leq D(l)/\sigma(l)$ we get the following recurrence relation:

$$D(l) \leq m\delta + \frac{\alpha\epsilon}{\beta} \sum_{h=1}^{l} (z^{(h)} - z^{(h-1)}) D(h-1) \tag{B.12}$$

Let η be the sequence defined by $\eta(0) = m\delta$ and for each $l \in [\tau]$:

$$\eta(l) = m\delta + \frac{\alpha\epsilon}{\beta} \sum_{h=1}^{l} (z^{(h)} - z^{(h-1)}) \eta(h-1) \tag{B.13}$$

By induction on l, we have $D(l) \leq \eta(l)$. Furthermore:

$$\eta(l) = m\delta + \frac{\alpha\epsilon}{\beta} \sum_{h=1}^{l-1} (z^{(h)} - z^{(h-1)}) \eta(h-1) + \frac{\alpha\epsilon}{\beta}(z^{(l)} - z^{(l-1)})\eta(l-1) \tag{B.14}$$

$$= (1 + \frac{\alpha\epsilon}{\beta}(z^{(l)} - z^{(l-1)}))\eta(l-1) \tag{B.15}$$

$$\leq \eta(l-1) \exp(\frac{\alpha\epsilon}{\beta}(z^{(l)} - z^{(l-1)})) \tag{B.16}$$

where the inequality comes from the inequality $1 + h \leq \exp(h)$ which holds for any $h \geq 0$. By induction on l, we get $\eta(l) \leq m\delta \exp(\frac{\alpha\epsilon}{\beta} z^{(l)})$. Hence $D(l) \leq m\delta \exp(\frac{\alpha\epsilon}{\beta} z^{(l)})$. By our stopping condition:

$$1 \leq D(\tau) \leq m\delta \exp(\frac{\alpha\epsilon}{\beta} z^{(\tau)}) \tag{B.17}$$

which directly leads to the claimed upper bound.

We now use this result to establish our lemma. Let γ be the ratio of the value of the optimal solution to the covering problem over the value of the scaled feasible primal solution returned by the algorithm. By substituting our bound on $\beta/z^{(\tau)}$ we get from Lemma 17:

$$\gamma = \frac{\beta \log_{1+\epsilon}(\frac{1+\epsilon}{\delta})}{z^{(\tau)}} \leq \frac{\alpha\epsilon \ln(\frac{1+\epsilon}{\delta})}{\ln(1+\epsilon)\ln((m\delta)^{-1})} \tag{B.18}$$

By choosing $\delta = (1+\epsilon)[(1+\epsilon)m]^{-1/\epsilon}$ we obtain $\ln((1+\epsilon)/\delta)/\ln((m\delta)^{-1}) = (1-\epsilon)^{-1}$ which leads to:

$$\gamma \leq \frac{\alpha\epsilon}{(1-\epsilon)\ln(1+\epsilon)} \leq \frac{\alpha\epsilon}{(1-\epsilon)(\epsilon - \frac{\epsilon^2}{2})} \leq \frac{\alpha}{(1-\epsilon)^2} \tag{B.19}$$

where the second inequality, which is valid for any $\epsilon \in]0, 1[$, comes from the Taylor series of $\ln(1+\epsilon)$. Hence, we choose ϵ such that $(1-\epsilon)^{-2} \leq 1+\omega$, so that the scaled solution is an $(1-\omega)/\alpha$ approximation to the packing problem, where ω is any desired accuracy. □

Notice that, if the subroutine is a PTAS (respectively FPTAS), we can choose $\alpha = 1+\epsilon$ as accuracy of the approximation algorithm. By setting $\delta = (1+\epsilon)[(1+\epsilon)m]^{-1/\epsilon^2}$ instead of the value used in the above proof, we get $\gamma \leq (1-\epsilon)^{-2} \leq 1+\omega$ which gives a PTAS (respectively FPTAS) for the packing problem.

B.1.4 Running time

The next lemma shows that the algorithm runs in time polynomial in the size of the input, provided the oracle runs in polynomial time.

Lemma 20. *The algorithm terminates after at most $O(m\lceil \log_{1+\epsilon}(\delta^{-1})\rceil)$ iterations. Hence, its overall complexity is $O(\omega^{-2} m \log(m) T_{Oracle})$, where T_{Oracle} is the time required to approximately solve the dual separation problem by calling the oracle, and ω is the desired accuracy.*

Proof. Initially $y_i^{(0)} = \delta/b_i$ for each i. The last time the value of the dual variable y_i was increased, its value just before the update was less than $1/b_i$, otherwise we could not have $D(\tau - 1) < 1$. Since the algorithm increases it by a factor at most $1 + \epsilon$, the final value of the dual variable y_i does not exceed $(1 + \epsilon)/b_i$. Each time one packing constraint is saturated (which happens at each iteration), the corresponding dual variable value is increased by a factor exactly $1 + \epsilon$. Thus, this can happen at most $\lceil \log_{1+\epsilon}(\frac{1}{\delta}) \rceil$ times for each dual variable. Since there are m dual variables, the total number of iterations is at most $m\lceil \log_{1+\epsilon}(\frac{1}{\delta}) \rceil$. From $\delta = (1 + \epsilon)[(1 + \epsilon)m]^{-1/\epsilon}$ combined with $\epsilon = 1 - (1 + \omega)^{-1/2}$, the number of iterations is upper-bounded by $O(\omega^{-2} m \log(m))$. The algorithm performs one call to the oracle at each iteration. The overall complexity follows. □

B.1.5 Algorithm

Algorithm 4 below provides the corresponding pseudo-code:

Algorithm 4 : the Garg-Könemann framework for packing problems

Input: An oracle to compute an α-approximate solution to the dual separation problem induced by matrix A and vector c, vector b, accuracy ω;
Output: An $(1-\omega)/\alpha$-approximated feasible solution x of value z to the packing problem;

1. Define $\epsilon = 1 - (1 + \omega)^{-1/2}$, $\delta = (1 + \epsilon)((1 + \epsilon)m)^{-1/\epsilon}$;
2. Initialize $y_i = \delta/b_i$, $x = 0$, $z = 0$;
3. **while** $b^T y < 1$ **do**
4. Call the oracle to find column q such that $A_q^T y/c_q \leq \alpha\sigma(y)$;
5. Compute row p which minimizes $b_i/A(i, q)$ over all rows $i \in I$;
6. Update primal vector x, $x_q = x_q + b_p/A(p, q)$ and primal value $z = z + c_q b_p/A(p, q)$;
7. Update dual vector y, $y_i = y_i(1 + \epsilon \frac{b_p A(i,q)}{b_i A(p,q)})$ for each row $i \in I$;
8. **end while**
9. Scale variables $x_j = x_j/\log_{1+\epsilon}(\frac{1+\epsilon}{\delta})$ for each $j \in J$ and value $z = z/\log_{1+\epsilon}(\frac{1+\epsilon}{\delta})$;
10. Return x and z.

B.2 The framework of Fleischer and Wayne

The presentation below is directly inspired from [22].

B.2.1 A reformulation of the dual problem

Given a dual variable vector (y, z), we define $D(y) = b^T y$ and $\sigma(z) = d^T z$. For any non-negative vector y we can define a new vector $z(y)$ by setting $z(y)_k = \min_{j \in C^k} \{A_j^T y\}$. Notice that vector $z(y)$ is non-negative provided that vector y is itself non-negative. In the following we use the notation $\sigma(y) = \sigma(z(y))$ and we focus on vector y. Let β be the optimal value of the following problem:

$$\beta = \min_{y \geq 0} \left\{ \frac{D(y)}{\sigma(y)} \right\} \tag{B.20}$$

The next lemma shows that this last problem is actually a reformulation of the dual of the max-min resource sharing problem.

Lemma 21. *Given an optimal solution (y^*, z^*) to the dual problem, we have $\beta = D(y^*)$.*

Proof. Let (y^*, z^*) be an optimal solution to the dual problem. We clearly have $A^T y^* \geq H^T z(y^*)$. Furthermore, $d^T z(y^*) \geq d^T z^* = 1$ thus $d^T z(y^*) > 0$. Without loss of generality, we can assume that $d^T z(y^*) = 1$ since otherwise we could just divide vector y^* by $d^T z(y^*)$ to get a new vector satisfying this assumption. We get $D(y^*) = \frac{D(y^*)}{\sigma(y^*)}$ so that $D(y^*) \geq \beta$. Now, let y be a non-negative vector such that $\beta = \frac{D(\mathrm{y})}{\sigma(\mathrm{y})}$. From the inequality $D(y^*) \geq \beta$ combined with the assumption on the feasibility of the dual problem, the quantity β is finite, hence $\sigma(\mathrm{y}) > 0$. We assume again that $\sigma(\mathrm{y}) = 1$, without loss of generality since we can scale vector y by $d^T z(\mathrm{y})$ so that this assumption is satisfied, while preserving the optimality of the scaled vector. Since $A^T \mathrm{y} \geq H^T z(\mathrm{y})$, the couple of vectors $(\mathrm{y}, z(\mathrm{y}))$ is a feasible solution to the dual problem with value $D(\mathrm{y}) = \frac{D(\mathrm{y})}{\sigma(\mathrm{y})} = \beta$. Thus $\beta \geq D(y^*)$. □

B.2.2 Framework of the algorithm

Assume we have an approximation algorithm which, given a dual variable vector y, returns a column q such that $A_q^T y \leq \alpha \min_{j \in J} \{A_j^T y\}$. We now describe how to incorporate

this approximation algorithm as a subroutine in a method to approximately solve the max-min resource sharing problem.

Again, we use two positive real numbers δ and ϵ whose values will be suitably chosen later. The algorithm works in phases, indexed by l, each phase consisting of as many iterations as the number of commodities. Hence, iterations are indexed by k. Start from primal vector $x^{(0)} = 0$ and dual vector $y_i^{(0)} = \delta/b_i$ for all i. We will fix the value of the variable λ at the end of the algorithm. At phase l, for each commodity k, we perform as many steps as required to distribute an amount d^k over a subset of columns of matrix A. More precisely, we start iteration k of phase l by setting the *remaining demand* ϱ^k to d^k. While ϱ^k remains positive, the algorithm first calls the oracle to find a column q such that $A_q^T y \leq \alpha \min_{j \in C^k}\{A_j^T y\}$ where y is the current dual vector. The algorithm then looks for a row p minimizing $b_i/A(i,q)$. Let $\nu = \min\{\varrho^k, b_p/A(p,q)\}$. The algorithm increases primal variable x_q by ν, decreases remaining demand ϱ^k by ν, and updates dual variable y_i for all i according to the following equation:

$$y_i = y_i \left(1 + \epsilon\nu \frac{A(i,q)}{b_i}\right) \tag{B.21}$$

Once ϱ^k is set to 0 we are done with iteration k. Phase l ends when we have taken care of all commodities. The procedure terminates as soon as $D(y) \geq 1$. In the sequel, we denote by τ the corresponding phase. At the end of this procedure, the algorithm performs one last call to the oracle for each commodity k, to compute column q^k such that $A_{q^k}^T y \leq \alpha \min_{j \in C_k}\{A_j^T y\}$. Define $z_k = A_{q^k}^T y/\alpha$ for each commodity k (although we define z_k at the end of each step, we only need to compute vector z at the end of the procedure). Scaling both y and z by $d^T z$ if this last quantity is smaller than 1, we obtain a feasible dual solution (y, z). In the first $\tau - 1$ phases, for each commodity k, the overall demand is $d^k(\tau - 1)$ since, at the end of each phase, the algorithm ensures $\varrho^k = 0$ for every k. Hence, at the end of the procedure $H_k x \geq d^k(\tau - 1)$ for every commodity k. Fix λ at $\min_{k \in K}\{H_k x/d_k\}$. From the previous remark, we have $\lambda \geq \tau - 1$. For brevity, we still use the notation $D(l) = D(y^{(l)})$ and $\sigma(l) = \sigma(y^{(l)})$ for each phase l. In order to get a feasible solution to the primal problem, we will scale the vector x returned by the algorithm and the value of variable λ by a suitable value, as emphasized by the next lemma.

Lemma 22. *Scaling down by a factor of* $\log_{1+\epsilon}(\frac{1}{\delta})$ *both the vector* x *returned by the previous algorithm and the value of variable* λ*, as defined above, gives a feasible solution to the max-min resource sharing problem.*

Proof. The argument is similar to the one used in the proof of Lemma 18. When the algorithm increases the value of variable x_q by $b_p/A(p,q)$, the left-hand-side of the i^{th} packing constraint (corresponding to the i^{th} row of matrix A) increases by $b_pA(i,q)/b_iA(p,q)$ which is less than one by definition of p. Each increase in the left-hand-side of the i^{th} packing constraint by 1 causes dual variable y_i to be multiplied by at least $1+\epsilon$. Since $D(\tau-1) < 1$, $y_i^{(\tau-1)} < 1/b_i$ for each i. From $y_i^{(0)} = \delta/b_i$, we get that the value of the left-hand-side of the i^{th} constraint after $\tau - 1$ phases is at most $b_i \log_{1+\epsilon}(1/\delta)$. This holds for all primal packing constraints, hence scaling down by $\log_{1+\epsilon}(1/\delta)$ the primal vector x, returned by the algorithm after $\tau - 1$ phases, gives a new vector satisfying those packing constraints. Scaling λ by the same value maintains the validity of the fairness constraints $d\lambda \leq Hx$, which concludes the proof. □

B.2.3 Approximation guarantee

The next lemma states that the feasible solution to the primal problem, obtained after scaling, has almost the same guarantee on its quality than the guarantee provided by the oracle used as a subroutine.

Lemma 23. *Given an oracle with ratio* α *to solve the dual separation problem for a fixed commodity, the proposed algorithm can be tuned to return a feasible solution to the primal problem whose value is no less than* $(1-\omega)/\alpha$ *times the optimal one, where* ω *is any given positive real number.*

Proof. Let α be the quality ratio guaranteed by the subroutine. Assuming $\beta \geq \alpha$ (we shall remove this assumption later), we show that $\beta/(\tau-1) \leq \alpha\epsilon/[(1-\epsilon)\ln((1-\epsilon)/(m\delta))]$. Let $y^{(l,k,s)}$ and $z^{(l,k,s)}$ be the value of the dual vectors y and z, respectively, at the end of step s in iteration k of phase l. Denote by $\nu^{(l,k,s)}$ the increment amount in this step (recall that $\nu = \min\{\varrho^k, b_p/A(p,q)\}$). Let also $y^{(l,k)}$ and $z^{(l,k)}$ be respectively the values of the dual

vectors y and z at the end of iteration k. At the end of step s, it holds that:

$$D(y^{(l,k,s)}) = \sum_{i \in I} b_i y_i^{(l,k,s)} \tag{B.22}$$

$$= \sum_{i \in I} b_i y_i^{(l,k,s-1)} + \epsilon \nu^{(l,k,s)} \sum_{i \in I} A_{iq^k} y_i^{(l,k,s-1)} \tag{B.23}$$

$$= D(y^{(l,k,s-1)}) + \epsilon \nu^{(l,k,s)} A_{q^k}^T y^{(l,k,s-1)} \tag{B.24}$$

$$= D(y^{(l,k,s-1)}) + \alpha \epsilon \nu^{(l,k,s)} z^{(l,k,s-1)} \tag{B.25}$$

Observe that y is monotonically increasing throughout any iteration and hence so is z. We get:

$$D(y^{(l,k,s)}) \leq D(y^{(l,k,s-1)}) + \alpha \epsilon \nu^{(l,k,s)} z^{(l,k)} \tag{B.26}$$

From $\sum_s \nu^{(l,k,s)} = d_k$ we obtain:

$$D(y^{(l,k)}) \leq D(y^{(l,k-1)}) + \alpha \epsilon d_k z^{(l,k)} \tag{B.27}$$

Summing over k in a phase l leads to:

$$D(l) \leq D(l-1) + \alpha \epsilon \sigma(l) \tag{B.28}$$

Recall that we assume $\beta \geq \alpha$. Since by definition $\beta \leq D(l)/\sigma(l)$, it comes for each phase l:

$$D(l) \leq \frac{D(l-1)}{1 - \alpha\epsilon/\beta} \tag{B.29}$$

From $D(0) = m\delta$ we deduce by induction on l:

$$D(l) \leq \frac{m\delta}{(1-\alpha\epsilon/\beta)^l} = \frac{m\delta}{(1-\alpha\epsilon/\beta)}(1 + \frac{\alpha\epsilon}{\beta - \alpha\epsilon})^{l-1} \leq \frac{m\delta}{(1-\epsilon)} \exp(\alpha\epsilon(l-1)/(\beta(1-\epsilon))) \tag{B.30}$$

where the last inequality comes from $\beta \geq \alpha$ (hence $\frac{1}{1-[\alpha\epsilon/\beta]} \leq \frac{1}{1-\epsilon}$) along with the Taylor series of $\exp(\frac{\alpha\epsilon}{\beta[1-\epsilon]})$. From our stopping condition we have:

$$1 \leq D(\tau) \leq \frac{m\delta}{(1-\epsilon)} \exp(\alpha\epsilon(\tau-1)/(\beta(1-\epsilon))) \tag{B.31}$$

which directly leads to:

$$\frac{\beta}{\tau - 1} \leq \frac{\alpha\epsilon}{(1-\epsilon)\ln((1-\epsilon)/(m\delta))} \tag{B.32}$$

which is the claimed upper bound.

We now use this result to establish our lemma. Let γ be the ratio of the value of the

optimal solution to the dual problem over the value of the scaled feasible solution returned by the algorithm. From Lemma 21 and Lemma 22 we have:

$$\gamma = \frac{\beta \log_{1+\epsilon}(1/\delta)}{\lambda} \leq \frac{\alpha\epsilon \ln(1/\delta)}{(1-\epsilon)\ln(1+\epsilon)\ln((1-\epsilon)/(m\delta))} \tag{B.33}$$

where the last inequality comes from $\lambda \geq \tau - 1$. For $\delta = (m/(1-\epsilon))^{-1/\epsilon}$ the ratio $\ln(1/\delta)/\ln((1-\epsilon)/(m\delta))$ equals $(1-\epsilon)^{-1}$ hence

$$\gamma \leq \frac{\alpha\epsilon}{(1-\epsilon)^2 \ln(1+\epsilon)} \leq \frac{\alpha\epsilon}{(1-\epsilon)^2(\epsilon - \epsilon^2/2)} \leq \frac{\alpha}{(1-\epsilon)^3} \tag{B.34}$$

where the second inequality comes from the Taylor series of $\ln(1+\epsilon)$. Hence, we choose ϵ such that $(1-\epsilon)^{-3} \leq 1+\omega$, so that the scaled solution is an $(1-\omega)/\alpha$ approximation to the max-min resource sharing problem, where ω is any fixed accuracy. □

B.2.4 Running time

We shall now explain how to remove the assumption $\beta \geq \alpha$. We use the same arguments as in [21, 22] which are based on an idea proposed in [135].

Lemma 24. *The algorithm runs for at most* $O(h\log(h)\lceil \log_{1+\epsilon}(m/(1-\epsilon))/\epsilon \rceil)$ *iterations.*

Proof. By weak duality combined with the feasibility of the primal solution returned by the proposed algorithm, we have:

$$1 \leq \gamma \leq \frac{\beta \log_{1+\epsilon}(1/\delta)}{\tau - 1} \tag{B.35}$$

where the last inequality comes from $\lambda \geq \tau - 1$. Hence the number of phases τ in the above procedure is less than $1 + \beta \log_{1+\epsilon}(1/\delta) = 1 + \beta \log_{1+\epsilon}(m/(1-\epsilon))/\epsilon$. Notice that this upper-bound depends on β which can be modified by scaling the instance vector d. Let ζ^k be the maximum sum of k-commodity variables satisfying all constraints $Ax \leq b$ when all variables corresponding to other commodities are set to 0. We also denote by x^k the corresponding vector and by x the aggregation of the vectors x^k over every commodity. Let $\zeta = \min_{k \in K} \zeta^k$. It should be clear from Sion's minimax theorem [136] that $\beta \leq \zeta$. Furthermore, the inequality $\zeta/h \leq \beta$ holds since vector x is a feasible solution to the

max-min resource sharing problem in the instance obtained by multiplying vector b by h. We scale demand vector d by $\frac{\zeta}{\alpha h}$ so as to get the new value of ζ equals to αh. Hence $\alpha \leq \beta \leq \alpha h$. If the algorithm does not stop within $\upsilon = 2\alpha\lceil \log_{1+\epsilon}(m/(1-\epsilon))/\epsilon \rceil$ phases then we know that $\beta > 2\alpha$. We then double the demand, hence β is halved but still at least α. The algorithm runs for an additional υ phases and if it does not halt we double the demand again. Since we halve the value of β after every υ phases, the total number of such phases is at most $\upsilon \log(h)$. There are at most h iterations per phase. The complexity follows. □

Observe that, although computing ζ may be intractable, we can compute an $(1-\omega)/\alpha$ approximation by combining the subroutine with the packing algorithm described above.

The next lemma shows that the algorithm runs in time polynomial in the size of the input, provided the oracle runs in polynomial time.

Lemma 25. *The number of steps in the entire algorithm exceeds the number of iterations by at most $O(m\lceil \log_{1+\epsilon}(m/(1-\epsilon))/\epsilon \rceil)$. Hence, its overall complexity is $O(\omega^{-2}(m + h\log(h))\log(m)T_{Oracle})$, where T_{Oracle} is the time required to approximately solve the dual separation problem by calling the oracle, and ω is the desired accuracy.*

Proof. We show how to bound the total number of steps. For each step but the last we have $\nu = b_p/A(p,q)$, hence the algorithm increases at least dual variable y_p by a multiplicative factor of $1+\epsilon$. From $D(\tau - 1) < 1$ we deduce that $y_i \leq 1/b_i$ for each i before the last step of the algorithm. Since y_i initially equals δ/b_i the total number of steps performed by the algorithm exceeds the number of iterations by at most $\lceil m \log_{1+\epsilon}(1/\delta) \rceil = \lceil m \log_{1+\epsilon}(m/(1-\epsilon))/\epsilon \rceil$. Combining this last upper-bound along with Lemma 24 and $\epsilon = 1 - (1+\omega)^{-1/3}$, the algorithm performs at most $O(\omega^{-2}(m + h\log(h))\log(m))$ steps. The oracle is called once at each step. The overall complexity follows. □

B.2.5 Algorithm

Algorithm 5 below provides the corresponding pseudo-code.

Algorithm 5 : the Garg-Könemann-Fleischer-Wayne method for max-min resource sharing problems

Input: An oracle to compute an α-approximate solution to the dual separation problem induced by matrices A and H, vector b, vector d, accuracy ω;

Output: An $(1-\omega)/\alpha$-approximated feasible solution x of value λ to the max-min resource sharing problem;

1. Define $\epsilon = 1 - (1+\omega)^{-1/3}$, $\delta = (m/(1-\epsilon))^{-1/\epsilon}$;
2. Initialize $y_i \leftarrow \delta/b_i$, $x \leftarrow 0$;
3. **while** $b^T y < 1$ **do**
4. $\quad \varrho \leftarrow d$;
5. $\quad L \leftarrow K$;
6. $\quad$ **while** $L \neq \emptyset$ and $b^T y < 1$ **do**
7. $\quad\quad$ Pick $k \in L$;
8. $\quad\quad L \leftarrow L \setminus \{k\}$;
9. $\quad\quad$ **while** $\varrho^k > 0$ and $b^T y < 1$ **do**
10. $\quad\quad\quad$ Call the oracle to find column q^k such that $A_{q^k}^T y \leq \alpha \min_{j \in C^k}\{A_j^T y\}$;
11. $\quad\quad\quad$ Compute row p^k which minimizes $b_i/A(i,q)$ over all rows $i \in I$;
12. $\quad\quad\quad$ Let $\nu = \min\{\varrho^k, b_{p^k}/A(p^k,q^k)\}$;
13. $\quad\quad\quad$ Update primal vector x, $x_q \leftarrow x_q + \nu$;
14. $\quad\quad\quad$ Update remaining demand vector ϱ, $\varrho^k \leftarrow \varrho^k - \nu$;
15. $\quad\quad\quad$ Update dual vector y, $y_i \leftarrow y_i(1 + \epsilon\nu\frac{A(i,q^k)}{b_i})$ for each row $i \in I$;
16. $\quad\quad$ **end while**
17. $\quad$ **end while**
18. **end while**
19. Scale variables $x_j \leftarrow x_j / \log_{1+\epsilon}(\frac{1}{\delta})$ for each known column $j \in J$;
20. Compute $\lambda \leftarrow \min_{k \in K}\{H_k x / d^k\}$;
21. Return x and λ.

Appendix C

Coding gains

C.1 Multicommodity coding gains

C.1.1 Multicommodity coding gain

Proof of Theorem 11:

Proof. The underlying idea of this theorem is again that routing using multicast only corresponds to the special case of using both multicast and network coding simultaneously in a network where no coding operation is performed but simple data duplication. Furthermore, the proof itself is very similar to the one of Theorem 3. Let $\mathbb{x}$ be a maximum weighted multicommodity Steiner flow. We shall build a feasible multicommodity coded flow $\mathbb{x}$ with the same value as $\mathbb{x}$. For each commodity k, each receiver $r \in R^k$ and each path p between s^k and r, define:

$$\mathbb{x}_p = \sum_{\substack{t \in \mathcal{T}^k \\ p_t^r = p}} \mathbb{x}_t \tag{C.1}$$

where for any commodity k, any Steiner tree $t \in \mathcal{T}^k$, and any receiver $r \in R^k$, p_t^r is the unique path between s^k and r in t. We first show that coded flow $\mathbb{x}$ satisfies all capacity

requirements assuming Steiner flow $\mathbb{x}$ does. Observe that for any arc a,

$$\sum_{k\in K}\max_{r\in R^k}\sum_{p\in\mathcal{P}_a^{(k,r)}}\mathbb{x}_p = \sum_{k\in K}\max_{r\in R^k}\sum_{p\in\mathcal{P}_a^{(k,r)}}\sum_{\substack{t\in\mathcal{T}^k\\ p_t^r=p}}\mathbb{x}_t \tag{C.2}$$

$$\leq \sum_{k\in K}\max_{r\in R^k}\sum_{t\in\mathcal{T}_a^k}\sum_{\substack{p\in\mathcal{P}^{(k,r)}\\ p_t^r=p}}\mathbb{x}_t \tag{C.3}$$

$$= \sum_{k\in K}\max_{r\in R^k}\sum_{t\in\mathcal{T}_a^k}\mathbb{x}_t \tag{C.4}$$

$$= \sum_{k\in K}\sum_{t\in\mathcal{T}_a^k}\mathbb{x}_t \tag{C.5}$$

$$= \sum_{t\in\mathcal{T}_a}\mathbb{x}_t \tag{C.6}$$

$$\leq c_a \tag{C.7}$$

where the first inequality stands since, if arc a is used by path p while p is part of Steiner tree t then a is also used by t. The second equality holds since, for any commodity k, there is only one unique path p_t^r between source s^k and receiver $r \in R^k$ in any Steiner tree $t \in \mathcal{T}^k$. The third equality stands because for any arc a, and any commodity k, the quantity $\sum_{t\in\mathcal{T}_a^k}\mathbb{x}_t$ is independent of the receiver. Finally recall that we assume $\mathbb{x}$ to be a feasible Steiner flow from which we get the last inequality. Thus coded flow $\mathbb{x}$ meets all capacity requirements. Regarding the value of $\mathbb{x}$, one has for any commodity k, and any receiver $r \in R^k$:

$$\sum_{p\in\mathcal{P}^{(k,r)}}\mathbb{x}_p = \sum_{p\in\mathcal{P}^{(k,r)}}\sum_{\substack{t\in\mathcal{T}^k\\ p_t^r=p}}\mathbb{x}_t \tag{C.8}$$

$$= \sum_{t\in\mathcal{T}^k}\sum_{\substack{p\in\mathcal{P}^{(k,r)}\\ p_t^r=p}}\mathbb{x}_t \tag{C.9}$$

$$= \sum_{t\in\mathcal{T}^k}\mathbb{x}_t \tag{C.10}$$

where, for any commodity k, the last equality comes again from the uniqueness of the path

p_t^r for a given Steiner tree $t \in \mathcal{T}^k$ and a given receiver $r \in R^k$. Hence we get:

$$\phi(\mathbb{x}) = \sum_{k \in K} w^k \min_{r \in R^k} \sum_{p \in \mathcal{P}^{(k,r)}} \mathbb{x}_p \tag{C.11}$$

$$= \sum_{k \in K} w^k \min_{r \in R^k} \sum_{t \in \mathcal{T}^k} \mathbb{x}_t \tag{C.12}$$

$$= \sum_{k \in K} w^k \sum_{t \in \mathcal{T}^k} \mathbb{x}_t \tag{C.13}$$

$$= \phi(\mathbb{x}) \tag{C.14}$$

By combining this last result with the feasibility of multicommodity coded flow $\mathbb{x}$, and the optimality of multicommodity Steiner flow $\mathbb{x}$, one get $\phi_S \leq \phi_C$, or equivalently $g_\phi \geq 1$. □

C.1.2 Concurrent coding gain

Proof of Theorem 12:

Proof. This proof is very similar to the one of Theorem 11 in that given a maximum concurrent multicommodity Steiner flow $\mathbb{x}$ we shall build a feasible concurrent multicommodity coded flow $\mathbb{x}$ with the same value as $\mathbb{x}$. Multicommodity coded flow $\mathbb{x}$ is defined according to Equation C.1 so that it satisfies all capacity requirements provided multicommodity Steiner flow $\mathbb{x}$ does. Furthermore, by definition of $\mathbb{x}$ the equality

$$\sum_{p \in \mathcal{P}^{(k,r)}} \mathbb{x}_p = \sum_{t \in \mathcal{T}^k} \mathbb{x}_t \tag{C.15}$$

still holds for any commodity k, and any receiver $r \in R^k$. Therefore, we get:

$$\lambda(\mathbb{x}) = \min_{k \in K} \min_{r \in R^k} \frac{1}{d^k} \sum_{p \in \mathcal{P}^{(k,r)}} \mathbb{x}_p \tag{C.16}$$

$$= \min_{k \in K} \min_{r \in R^k} \frac{1}{d^k} \sum_{t \in \mathcal{T}^k} \mathbb{x}_t \tag{C.17}$$

$$= \min_{k \in K} \frac{1}{d^k} \sum_{t \in \mathcal{T}^k} \mathbb{x}_t \tag{C.18}$$

$$= \lambda(\mathbb{x}) \tag{C.19}$$

By combining this last result with the feasibility of multicommodity coded flow $\mathbb{x}$, and the optimality of multicommodity Steiner flow $\mathbb{x}$, one get $\lambda_S \leq \lambda_C$, or equivalently $g_\lambda \geq 1$. □

C.1.3 Cost coding gain

Proof of Theorem 18:

Proof. Let $\mathbb{x}$ be a minimum cost Steiner flow. We shall build a feasible coded flow $\mathbb{y}$ with a cost not greater than the one induced by $\mathbb{x}$. For each receiver r and each path p from s to r, we define:

$$\mathbb{y}_p = \sum_{\substack{t \in \mathcal{T} \\ p_t^r = p}} \mathbb{x}_t \tag{C.20}$$

where p_t^r is the unique path in Steiner tree t from the source s to the receiver r. As previously, it can be shown that $\mathbb{y}$ is a coded flow whose nominal throughput between the source and any receiver is exactly the one provided by $\mathbb{x}$ to all receivers. More precisely, we have for each receiver r:

$$\sum_{p \in \mathcal{P}^r} \mathbb{y}_p = \sum_{t \in \mathcal{T}} \mathbb{x}_t = d \tag{C.21}$$

So it remains to show that the cost of $\mathbb{y}$ on any arc a is not greater than the one of $\mathbb{x}$. For each arc a and each receiver r we have again:

$$\sum_{p \in \mathcal{P}_a^r} \mathbb{y}_p \leq \sum_{t \in \mathcal{T}_a} \mathbb{x}_t \tag{C.22}$$

so that for each arc a:

$$\max_{r \in R} \sum_{p \in \mathcal{P}_a^r} \mathbb{y}_p \leq \sum_{t \in \mathcal{T}_a} \mathbb{x}_t \tag{C.23}$$

which directly implies the expected result by monotonicity of the function ψ_a of each arc a. Hence the overall cost induced by the coded flow $\mathbb{y}$ is smaller or equal to the one induced by the Steiner flow $\mathbb{x}$. By optimality of $\mathbb{x}$ we get $\psi_C \leq \psi_S$, or equivalently $g_\psi \geq 1$. □

C.1.4 Survivable cost coding gain

Proof of Theorem 26:

Proof. Let $\mathbb{x}$ be a minimum cost survivable Steiner flow. We shall build a survivable coded flow $\mathbb{y}$ with a cost not greater than the one induced by $\mathbb{x}$. For each receiver r and each

path p from s to r, we define again:

$$\chi_p = \sum_{\substack{t \in \mathcal{T} \\ p_t^r = p}} \aleph_t \tag{C.24}$$

where p_t^r is the unique path in Steiner tree t from the source s to the receiver r. We already proved that χ is a coded flow whose nominal throughput between the source and any receiver is exactly the one provided by $\aleph$ to all receivers. More precisely, we have for each receiver r:

$$\sum_{p \in \mathcal{P}^r} \chi_p = \sum_{t \in \mathcal{T}} \aleph_t \tag{C.25}$$

Recall from the study of the cost coding gain that for each arc a and each receiver r:

$$\sum_{p \in \mathcal{P}_a^r} \chi_p = \sum_{t \in \mathcal{T}_a^r} \aleph_t \tag{C.26}$$

Once combined, Equation (C.25) and Equation (C.26) imply that for each arc a and each receiver r:

$$\begin{aligned}
\sum_{p \in \mathcal{P}^r \setminus \mathcal{P}_a^r} \chi_p &= \sum_{p \in \mathcal{P}^r} \chi_p - \sum_{p \in \mathcal{P}_a^r} \chi_p && \text{(C.27)} \\
&= \sum_{t \in \mathcal{T}} \aleph_t - \sum_{t \in \mathcal{T}_a^r} \aleph_t && \text{(C.28)} \\
&= \sum_{t \in \mathcal{T} \setminus \mathcal{T}_a^r} \aleph_t && \text{(C.29)} \\
&\geq d && \text{(C.30)}
\end{aligned}$$

where the last inequality comes from the feasibility of Steiner flow $\aleph$. Thus, coded flow χ satisfies all survivability requirements. Now, let $\hbar$ be the arc projection of coded flow χ as defined in Equation (5.56). Combining the definition of $\hbar$ with Equation (C.26) one gets for each arc a and each receiver r:

$$\begin{aligned}
\hbar_a &= \max_{r \in R} \sum_{p \in \mathcal{P}_a^r} \chi_p && \text{(C.31)} \\
&= \max_{r \in R} \sum_{t \in \mathcal{T}_a^r} \aleph_t && \text{(C.32)} \\
&\leq \sum_{t \in \mathcal{T}_a} \aleph_t && \text{(C.33)}
\end{aligned}$$

where the last inequality comes from the inclusion $\mathcal{T}_a^r \subseteq \mathcal{T}_a$. Finally observe that, for each arc a, the inequality:

$$\hbar_a \leq \sum_{t \in \mathcal{T}_a} \varkappa_t \tag{C.34}$$

along with the non-negativity of each weight w_a, directly imply that the overall cost induced by coded flow χ is not greater than the one induced by Steiner flow $\varkappa$. By optimality of $\varkappa$ we deduce $\chi_C \leq \chi_S$, or equivalently $g_\chi \geq 1$. □

C.1.5 Survivable coding gain

Proof of Theorem 30:

Proof. The proof follows the same line of reasoning as the one of Theorem 26 above. □

le cnam

Thibaut LEFEBVRE

Optimization of information flows in telecommunication networks

orange™

Résumé :
Dans les réseaux de télécommunications, la demande croissante pour de nouveaux services, comme la diffusion de vidéos en continu ou les conférences en ligne, engendre un besoin pour des dispositifs de télécommunication où le même contenu est acheminé depuis un émetteur unique vers un groupe de récepteurs. Cette évolution ouvre la voie au développement de nouvelles techniques d'acheminement des données, comme le multicast qui laisse un nœud du réseau copier ses données d'entrée puis retransmettre ces copies, ou le codage réseau, qui est une technique permettant à un nœud d'effectuer des opérations de codage à partir de ses données d'entrée. Cette thèse traite de la mise en place de techniques de codage au sein d'un réseau multicast filaire. Nous formalisons certains problèmes qui apparaissent naturellement dans ce contexte grâce à la recherche opérationnelle et à des outils d'optimisation mathématique. Notre objectif est de développer des modèles et des algorithmes afin de calculer, au moins de manière approchée, certaines grandeurs qui ont vocation à être pertinentes dans le cadre de la comparaison de techniques d'acheminement de données dans un réseau de télécommunications. Nous évaluons ainsi, d'un point de vue à la fois théorique et expérimental, l'impact induit par l'introduction de techniques de codage au sein d'un réseau multicast. Nous nous concentrons en particulier sur des critères importants pour un opérateur de télécommunication, comme la maximisation du débit d'information entre une source et un ensemble de destinataires dans le réseau, la minimisation de la congestion sous contrainte de demande, ou la minimisation de la perte de débit ou du coût induit par l'acheminement des données dans un réseau soumis à des pannes.

Mots clés :
Optimisation réseau, Programmation mathématique, Codage réseau, Multicast, Multiflot, Arbre de Steiner

Abstract:
In telecommunication networks, the increasing demand for new services, like video-streaming or teleconferencing, along with the now common situation where the same content is simultaneously requested by a huge number of users, stress the need for point to many data transmission protocols where one sender wishes to transmit the same data to a set of receivers. This evolution leads to the development of new routing techniques like multicast, where any node of the network can copy its received data and then send these copies, or network coding, which is a technique allowing any node to perform coding operations on its data. This thesis deals with the implementation of coding techniques in a wired multicast network. We formalize some problems naturally arising in this setting by using operations research and mathematical optimization tools. Our objective is to develop models and algorithms which could compute, at least approximately, some quantities whose purpose is to be relevant as far as forwarding data using either multicast and network coding in telecommunications networks is concerned. We hence evaluate, both in theory and numerically, the impact of introducing coding techniques in a multicast network. We specifically investigate relevant criteria, with respect to the field of telecommunications, like the maximum amount of information one can expect to convey from a source to a set of receivers through the network, the minimum congestion one can guarantee while satisfying a given demand, or the minimum loss in throughput or cost induced by a survivable routing in a network prone to failures.

Keywords:
Network optimization, Mathematical programming, Network coding, Multicast, Multicommodity flow, Steiner tree