# Décomposition de multi-flots et localisation de caches dans les réseaux

Pierre-Olivier Bauguion




**Télécom SudParis**

ÉCOLE DOCTORALE EDITE

# THÈSE

n° 2014TELE0010

préparée à **SAMOVAR** et à **Orange Labs**

présentée par

**Pierre-Olivier BAUGUION**

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ
Spécialité : INFORMATIQUE

# Décomposition de multi-flots et localisation de caches dans les réseaux

Soutenue publiquement le 22 septembre 2014 devant le jury :

| | | | |
|---|---|---|---|
| D. | Nace | Rapporteur | Université de Technologie de Compiègne, France |
| V. | Gabrel | Rapporteur | Université Paris-Dauphine, France |
| P. | Chrétienne | Examinateur | Université Pierre et Marie Curie, France |
| M. | Labbé | Examinateur | Université Libre de Bruxelles, Belgique |
| M. | Didi Biha | Examinateur | Université de Caen-Basse Normandie, France |
| W. | Ben Ameur | Directeur de thèse | Télécom SudParis, France |
| E. | Gourdin | Co-encadrant | Orange Labs, Issy-les-Moulineaux, France |

# Table des matières

*Si je pouvais t'offrir le bleu secret du ciel*
*Brodé de lumière d'or et de reflets d'argents*
*Le mystérieux secret, le secret éternel*
*De la nuit et du jour, de la vie et du temps*

*Avec tout mon amour je le mettrais à tes pieds*
*Mais tu sais je suis pauvre et je n'ai que mes rêves*
*Alors c'est de mes rêves qu'il faut te contenter*
*Alors marche doucement, car tu marches sur mes rêves*

W.B. Yeats - traduction

# Remerciements

A l'instar de l'optimisation la thèse est pour moi un voyage d'apprentissage dans l'espace des connaissances. C'est un chemin qui se construit pour son objectif, s'appuie sur les étapes déjà atteintes, et se réoriente par les obstacles qu'il rencontre. Si nous nous satisfaisons d'atteindre les frontières de cet espace lorsque nous recherchons l'optimum, la thèse, elle, se distingue par son ambition de les repousser. Essayer de révéler un nouvel espace, même infinitésimal, le tout au prix d'un effort parfois considérable.

Or, un voyage dans l'inconnu nécessite de bons guides. Je remercie chaleureusement mes encadrants Walid Ben Ameur et Eric Gourdin pour m'avoir fait confiance, m'avoir partagé l'excellence de leurs savoirs et compétences avec pédagogie, ainsi que pour leur sympathie et leur patience ; travailler avec leur encadrement était un privilège.

Un voyage possède une destination. Je remercie Virginie Gabrel et Dritan Nace, d'avoir accepté d'être les rapporteurs de mon travail de thèse, et Philippe Chrétienne, Mohammed Didi Biha et Martine Labbé d'en être les examinateurs.

Un voyage nécessite aides et moyens. Je remercie Orange, et plus particulièrement Nabil pour l'accueil au sein de son équipe et Adam, dans son OR/TC. Je remercie l'équipe elle-même, que j'ai appris à connaître et apprécier. Christian mon super co-bureau qui m'a soutenu (moi et ma tour) jusqu'au bout sans faillir. Yaya, Nan'cy, et Philippe mes « supers-copains », avec qui partager un moment convivialo-pausal au matin était un plaisir incontournable. Olivier et Matthieu qui m'ont encadré durant mon stage qui précéda ma thèse. Et enfin les plus ou moins assidus de la pause de l'après-midi, Alain, Bruno, Florence et Luca. Je remercie également mes nouveaux collègues de MIC, parmi notamment Fabien, Jorge, Lionel, Manel, Sabine et Sonia pour leur soutien lors de la dernière ligne droite.

Un voyage induit des rencontres. J'ai eu la chance de (presque) partager la « grande promo » des thésards d'Orange, de partager avec eux des moments forts avec eux ainsi qu'avec leurs conjoints, qui sont désormais des amis. Amel, Ghida, Le Blog, Jean-Bobin, Max et Ralu. J'ai largement bénéficié de leur soutien. Je remercie aussi les autres thésards post-docs et les stagiaires que j'ai pu rencontrer.

Un voyage requiert des bons compagnons. Benoît, instigateur entre deux parties de DOTA des « JM » et des « c'est quoi c'est [...]» qu'on finit par utiliser à notre grand désarroi. Je n'oublierai pas les « surges de Bauguionnerie » que j'ai infligées jusqu'au bout à Cédric, et qui les a reçues avec philosophie. Je le remercie, d'avoir été mon compagnon d'infortune, celui qui parle vrai et qui m'a appris qu'il y avait grande sagesse dans la phrase « si ça ne marche pas essaie une autre solution ». Le tout avec cette conviction inébranlable devant laquelle seule sa dulcinée, Perrine,

semble rester hermétique. David, dont l'amitié brute mais sincère me fera toujours sourire, mais pas autant que sa chance légendaire. 3xt, le papa de la troupe, qui ne se départ jamais de sa nonchalance. Hugo, le seul que j'ai réussi à entraîner dans mes lubies de CCG/LCG, et qui je n'entends plus rager devant son PC depuis qu'il a rencontré Charlotte. Lucie qui doit apprendre un jeu de plateau tous les mois, et Seb qui partage cette passion. J'espère qu'il ne m'en veut pas trop d'avoir réécrit la guerre froide. Je remercie Alex avec qui j'ai grandi depuis le collège, et Dimitri qui nous a rejoint par la suite. Trystan mon plus vieil ami d'enfance avec qui je partage une conversation ininterrompue, et qui semble toujours croire en moi. Wawa, avec qui je forme une équipe de choc en "coop", et qui m'apprend toujours à relativiser. Et tous les autres, qui sans les nommer, je n'oublie pas.

Je remercie mes oncles, tantes, cousins, cousines et grand-parents, qui malgré le temps et l'espace qui séparent nos rencontres me sont chers. A Louis, qui m'accompagne et m'enseigne toujours d'une certaine façon, et à Lisette que je n'ai jamais eu la chance de connaître.

Un voyage se construit par une motivation. L'une des forces principales qui me permet d'avancer et de subir les intempéries, tout autant que de passer d'excellents moments, mes sœurs et leurs conjoints la délivrent. Merci Aurélie, Ben, Lucile et Thom, la foi que vous placez en moi, votre soutien et votre amour me sont indispensables. Les babillages et la frimousse de la petite "Pikachunette" ont également su me décrocher des rires et des sourires à des moments clés(-pooch).

Enfin, tout voyage possède une origine. A mes parents, dont la liste des choses pour lesquelles j'ai à les remercier est trop longue pour être couchée sur cette page. J'espère que ce qu'ils ont pu recevoir de leurs enfants est à la hauteur des efforts et des sacrifices qu'ils ont faits.

Merci.

A la mémoire d'Alberto,

*Ils ont un drapeau noir*
*En berne sur l'Espoir*
*Et la mélancolie*
*Pour traîner dans la vie*
*Des couteaux pour trancher*
*Le pain de l'Amitié*
*Et des armes rouillées*
*Pour ne pas oublier*

Leo Ferré, Les anarchistes

*"Either I make it down there in one piece and I have one hell of a story to tell, or I burn up in the next ten minutes. Either way, whichever way, no harm, no foul!*

*Because either way, it'll be one hell of a ride. I'm ready...."*

Gravity—Alfonso Cuaron, monologue du Docteur Ryan Stone

# Résumé

Les nouveaux acteurs, les nouveaux services et les nouveaux contenus multimédias qui transitent sur le réseau internet génèrent un trafic et des débits de plus en plus élevés. Ceci peut occasionner une congestion, source de latence et de dépréciation de la qualité de service ressentie par les utilisateurs. Un fournisseur d'accès à Internet dont l'objectif est de garantir un réseau d'excellence doit donc prendre des mesures pour améliorer sans cesse la fluidité de son réseau. Cela passe notamment par la mise en place d'un réseau de distribution de contenus (déploiement de dispositifs sur le réseau existant).

Dans un premier temps cette thèse présente des approches de programmation dynamique de localisation de serveurs optimales dans des arborescences. Nous présentons également un approche pour résoudre le problème de déploiement de CDN et de $k$ serveurs à l'aide de l'algorithme exact et polynomial d'intersection de matroïdes. Nous explicitons ensuite ce qu'est un cache et quelles sont ses caractéristiques. Nous définissons ensuite les hypothèses effectuées et la modélisation associée pour le déploiement de caches transparents dans une arborescence, et le lien avec les algorithmes existants présentés précédemment. Nous présentons alors un modèle complet pour un programme linéaire en nombres entiers (PLNE) et un nouveau paradigme de programmation dynamique pour résoudre ce même problème. Nous montrons alors en quoi cette approche se généralise à des problèmes connexes de localisation dans les arborescences, ainsi que ses performances.

D'un point de vue plus théorique, nous mesurons la capacité d'un réseau donné par le routage optimal de ses demandes, et, de ce fait, ses liens critiques. Nous manipulons alors le problème de flot concurrent maximal (FCM), un problème classique de la littérature de recherche opérationnelle. Nous exhibons alors de nouvelles formulations exactes pour résoudre ce problème, ainsi que les problèmes de multi-flots de manière plus générale. Une heuristique de construction de formulation pour le FCM est également proposée pour tirer parti de la distribution spécifique des capacités d'une instance. Nous montrons alors la supériorité des performances de ces nouvelles formulations par le biais de comparaisons.

Enfin, nous décrivons un algorithme exact et fortement polynomial pour résoudre le problème de flot concurrent maximal dans le cas d'une seule source. Nous montrons l'efficacité pratique d'une telle approche, comparée aux meilleures formulations explicitées précédemment.

# Abstract

Streaming requirements on Internet network are more driven by new actors, new services and new digital contents. This leads to high probability of congestion and latency, and therefore, a critical decrease of quality of service and/or experience for customers. An internet service provider (ISP) whose goal is to guarantee a first-class performance, needs to take measures to constantly enhance the fluidity of the traffic streaming on its network. One way to face the problem is to build a Content Delivery Network (CDN). A CDN mainly consists in the deployment of different devices on an existing network.

First of all, this thesis presents dynamic programming approaches to tackle server location problems in tree networks. Then, we address a variation of the matroid intersection algorithm to solve the $k$-server/cache location problem. We start by giving the definition and characteristics of transparent-caching, as well as the hypothesis that we will use it to build models for transparent cache location in tree network. We tract it into a Mixed Integer Program, and formulate a new paradigm of dynamic programming. We show the relevance of such approach for our problem, and to what extent it can be tractable in other related problems.

From a more theoretical point of view, we manage to measure the capacity of a network which is given by the optimal routing strategy, and hence, to identify its critical links. We deal with the Maximum Concurrent Flow (MCF), a classical combinatorial optimization problem. We propose new models and formulations to solve this problem exactly, and more general multi-flows problems as well. A heuristic is also given, to adapt the model to the specific instance values. We experiment these formulations to show the improvements they can provide. Finally, we describe the first strongly polynomial algorithm to solve the maximum concurrent flow to optimality in the single source case. We show the efficiency of such an approach, even compared to the best models previously presented.

# Première partie

# Localisation de caches

# Chapitre 1

# Optimisation de déploiement de réseau de distribution de contenus

## Introduction

Dans ce chapitre, nous allons présenter quelques éléments techniques fondamentaux des réseaux de distribution de contenus (CDN) et des caches transparents. Nous verrons alors que des problématiques de localisation à la fois de dispositifs et de contenus émergent dans le déploiement d'un CDN. Parmi la littérature qui traite ces problématiques, nous nous pencherons plus particulièrement sur deux études existantes de localisation de caches et/ou de serveurs dans des réseaux arborescents et d'observer quelles sont les similitudes avec la description du problème que nous allons traiter dans le chapitre 3. Ensuite, nous verrons en quoi ces deux approches de programmation dynamique proposées pour résoudre ce type de problèmes diffèrent, aussi bien dans le détail de leurs fonctionnements que dans leurs forces et leurs limites. Ceci nous permettra de mieux comprendre la nécessité d'étendre ces approches pour pouvoir aborder les spécificités de notre problème.

## 1.1 Les réseaux de distribution de contenus

Dans les années soixante, les Etats-Unis élaborent l'un des tout premiers réseaux longue distance, qui relie des ordinateurs entre eux et leur permet d'échanger des informations de nature digitale. Cette innovation marquera le point de départ vers des réseaux de taille de plus en plus grande, aux technologies standardisées (notamment au niveau des protocoles), et aux acteurs plus variés. Réseau de réseaux mondial, dépourvu de centre névralgique, l'Internet commence alors à naître. L'apparition d'organismes, tels que les FAI, (Fournisseur d'Accès à Internet) conjointement au développement et à la production de masse des composants informatiques (terminaux), démocratisent alors l'accès à ce réseau pour les particuliers dans les années 1990. L'Internet s'invite alors dans le quotidien des foyers, que ce soit à travers le World Wide Web, la messagerie instantanée ou le courrier électronique. S'ensuivra la naissance d'une économie dématérialisée, qui attirera de nombreux investissements pour de nouveaux services s'appuyant sur ce réseau du futur. Afin de faire face au succès de l'Internet et au trafic croissant qu'il générait, les réseaux ont subi de sensibles améliorations technologiques afin de satisfaire la demande et augmenter la qualité de service (débit garanti, temps de latence...).

Aujourd'hui, les contenus multimédias numériques se multiplient et requièrent de plus en plus de mémoire de stockage (par exemple les vidéos haute définition). Parallèlement, des services proposent de distribuer ces contenus au travers de l'Internet (YouTube, vidéos à la demande, catch-up TV), ce qui occasionne un trafic de grand volume. Afin de fournir ces contenus volumineux à des clients nombreux et épars, il est alors généralement nécessaire de construire une architecture coopérative de serveurs placés directement sur le réseau Internet : c'est ce qu'on appelle un réseau de distribution de contenus (Content Delivery Network) dont la figure (1.1) montre un exemple comparé à une solution "traditionnelle".

**Définition 1.** *Un réseau de distribution de contenus est constitué et caractérisé par :*

1. *Le trafic considéré ; les fichiers qui le concernent ("catalogue" de fichiers)*
2. *Les serveurs centraux sur le réseau, qui rassemblent et peuvent fournir la totalité du catalogue*
3. *Les serveurs d'appoint sur le réseau (parfois nommés réplicats) destinés à servir un ensemble de clients localement plus proches sur le réseau*
4. *Du service/intelligence qui régit la cohésion de tous ces éléments, et en optimise le fonctionnement*

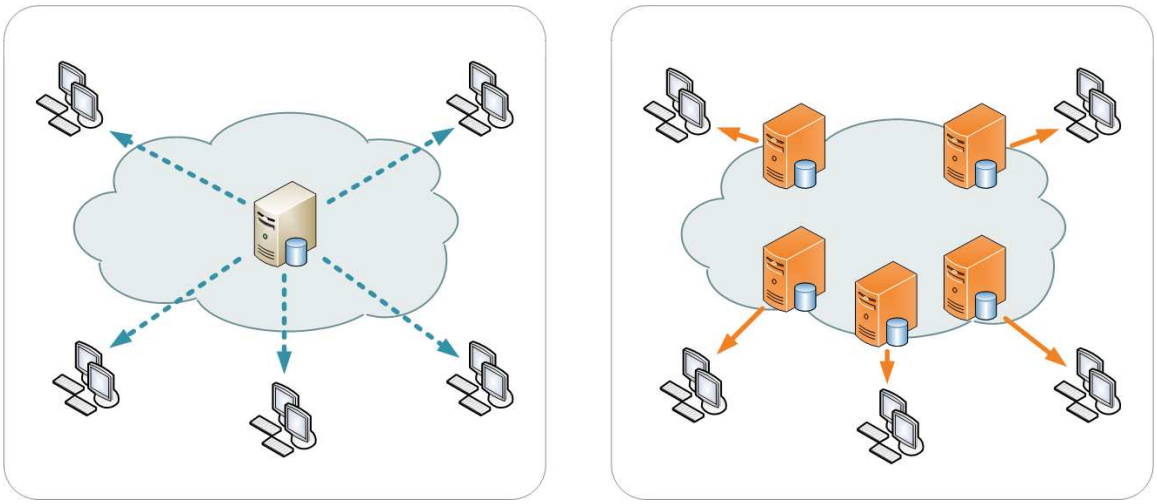


FIGURE 1.1 – *Une solution de stockage traditionnelle avec serveur central uniquement (à gauche) ; une architecture CDN (à droite)*

Si les fournisseurs de contenus, comme YouTube ou TF1, ont un intérêt à améliorer la diffusion de leurs contenus, et donc à déployer un CDN, ce peut également être le cas d'un fournisseur d'accès. Celui-ci, outre les services internes qu'il peut proposer, doit assurer le bon fonctionnement du réseau qu'il contrôle et offre à ses clients. La qualité de service et/ou expérience passe par des indicateurs tels que la latence ou le débit constaté, et constitue la valeur ajoutée même que fournit un opérateur tel qu'Orange. Dans ce dernier cas, le CDN est alors un moyen de soulager le trafic sur tout ou partie de son réseau. Le CDN peut alors être différencié (spécifique à un type de trafic), ou partagé (indifférencié au type de trafic). Dans tous les cas, la réflexion sur le déploiement d'un CDN est primordiale pour en maximiser l'efficacité à moindre coût.

L'architecture d'un CDN consiste plus précisément en la gestion de la distribution des contenus du serveurs central vers les serveurs d'appoint (des "clones" partiels du serveur central), du positionnement de ces derniers sur le réseau (plus avancés vers un groupe de clients), et enfin, de la politique de routage des requêtes clients et des contenus (la coopération entre les différents dispositifs pour satisfaire les demandes). Si le choix de la localisation des serveurs d'appoint est une décision de long terme, ce n'est pas nécessairement le cas des contenus et du routage des requêtes. En effet, les contenus et leurs popularités ont tendance à changer rapidement (effets de "buzz" par exemple), d'où l'intérêt pour un serveur central —et à plus forte raison un réplicat de serveur—, de changer son catalogue de contenus. Tous ces comportements dynamiques sont très difficiles à prédire, à décrire et à modéliser.

La théorie des graphes a pu permettre de générer des approches pour déterminer les meilleures (ou, le cas échéant de bonnes) décisions pour déployer ces serveurs d'appoint à moindre coût. La littérature de recherche opérationnelle parle alors plus généralement de problèmes de localisation de dépôts (ou "facility location" en anglais) où l'objectif est d'optimiser le placement de dispositifs dans un graphe dans le but de minimiser (ou maximiser) une fonction objectif. Dans le cas d'un déploiement de CDN, le dépôt est alors assimilable au serveur, tandis que la marchandise représente l'information digitale.

### 1.1.1 Coûts linéaires et coûts fixes

Parmi les cas les plus simples de ces problématiques de localisation, il existe le problème de placement de dépôts et d'acheminement de marchandises à moindre coût, où le coût d'installation des dépôts et l'utilisation des arcs du réseaux sont uniquement dépendants (par une relation linéaire de coût) de leur utilisation. La contrainte unique est de fournir les clients du réseau depuis un des dépôts pour répondre à leurs demandes en marchandises.

De manière plus formelle, nous pouvons nous munir d'un graphe $G = (V, A)$, d'arcs décrits par l'ensemble $A$ et de sommets décrits par l'ensemble $V$, et nous cherchons à minimiser le coût de transmissions des marchandises depuis un dépôt vers chaque client $k$ ($k \in V$), de demande $D_k$. On introduit les variables $f_a, a \in A$ et $y_i, \forall i \in V$ explicitant respectivement le flot de marchandises sur l'arc $a$, et l'installation d'un dépôt sur le nœud $i$.

Ceci forme le modèle suivant :

$$\text{Localisation de dépôt 1} \begin{cases} \min \sum_{i \in V} C_c.y_i + \sum_{a \in A} w_a.f_a & (1.1) \\ \sum_{a \in \delta^+(i)} f_a - \sum_{a \in \delta^-(i)} f_a = D_i - y_i, \qquad \forall i \in V & (1.2) \\ f_a, y_i \geq 0, \qquad \forall a \in A, \forall i \in V & (1.3) \\ & (1.4) \end{cases}$$

Le modèle minimise (1.1), les coûts linéaires d'installation de serveur $C_c$ et des transmissions $w_a, \forall a \in A$. La contrainte (1.2) permet d'assurer les lois de conservations de flots tout en assurant la satisfaction des demandes, et l'envoi éventuel de marchandises depuis un dépôt installé.

Les arcs ne possédant pas de capacités, l'approvisionnement total d'un client peut être effectué par un chemin unique depuis un dépôt choisi, et pour un coût uniquement dépendant de

la somme des coûts linéaires de son sommet de départ (coût linéaire d'installation du dépôt) et des liens qu'il parcourt. Le choix du chemin le moins cher, ne dépend donc pas de la valeur de la demande. Un même dépôt peut également alimenter également plusieurs clients, en payant les coûts de la même façon. En réalité, on peut ramener ce problème à un problème d'arborescence des plus courts chemins, par un simple ajout de sommet "d'origine" virtuel comme nous l'illustre la figure 1.2. Ce problème polynomial peut être résolu par des algorithmes fortement polynomiaux (par exemple un algorithme de Dijkstra avec une complexité de $O(|V|^2)$).

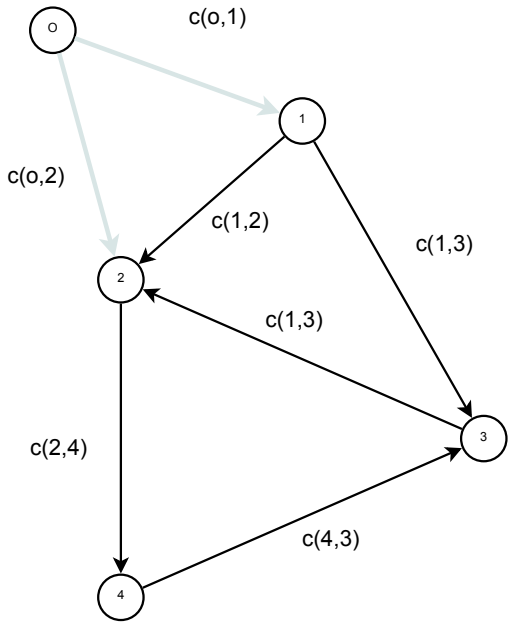



FIGURE 1.2 – *Graphe réel constitué des sommets* $\{1, 2, 3, 4\}$*, et de leurs liaisons. Le sommet de "départ" virtuel est noté en o, et les arcs successeurs sont les arcs modélisant l'installation de serveur et sont représentés en gris.*

Un autre problème simple, consiste à placer des dépôts et des liaisons à moindre coût, mais dont les coûts d'installation et d'utilisation des arcs du réseaux sont fixes dans les deux cas. La création d'un lien étant souvent valable dans les deux sens (la création d'une route génère, par exemple, est utilisable dans les deux sens), les liens sont alors représentés par des arêtes (et non des arcs).

De manière analogue au problème précédent, nous pouvons alors modéliser le problème comme un arbre couvrant de poids minimum. De nouveau, ce problème est polynomial et peut être résolu par un algorithme de Prim ou Kruskal en une complexité de $O(|E|)$ ($E$ représentant le nombre d'arêtes éligibles à l'installation d'un lien). Le chapitre suivant présente de manière plus précise ce problème ainsi que sa version du $k$-médian qui nous intéressera plus particulièrement.

### 1.1.2 Problèmes de localisation difficiles

Dans de nombreux problèmes, les coûts d'installation sont fixes (grandeur intensive), alors que les coûts d'acheminement dépendent de la quantité acheminée (grandeur extensive). Il est alors intéressant de remarquer que la combinaison de ces différents types de coûts définit un nouveau problème qui lui, est difficile (au sens de la complexité mathématique). C'est le problème que l'on connait sous la vocable de "problème de localisation de dépôts sans capacité" ("Uncapacitated Facility Location Problem" en anglais parfois abrégé en "UFLP") dont la figure 1.3 nous montre une exemple de solution.

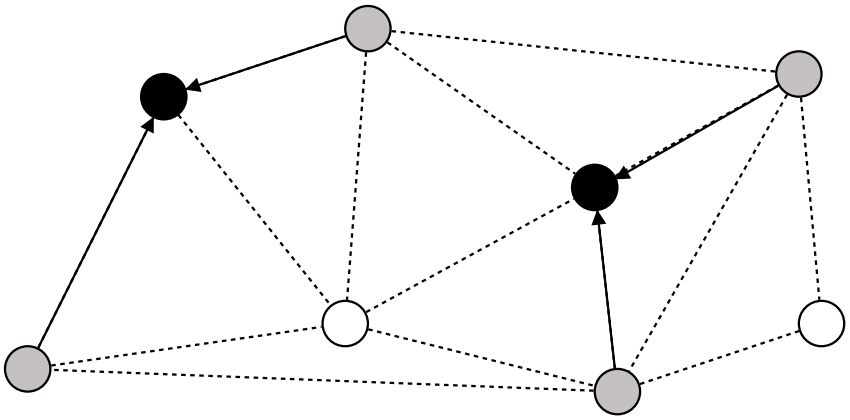


FIGURE 1.3 – *Les "facility" (en noir), placés sur les endroits éligibles (en blanc), où s'approvisionnent les clients (en gris).*

Regardons un modèle qui explicite ce problème. Nous nous munissons d'un graphe $G = (V, A)$, d'arcs $A$ et de sommets $V$, et nous cherchons à minimiser le coût de transmissions des marchandises depuis un dépôt vers chaque client $i$ ($i \in V$), de demande $D_i$, et d'installation de dépôts. On introduit les variables $f_a, a \in A$ et $y_i, \forall i \in V$ explicitant respectivement le flot de marchandises sur l'arc $a$, et les variables d'installation d'un dépôt sur le nœud $i$.

$$\text{"UFLP"} \begin{cases} \min \sum_{i \in V} C_c.y_i + \sum_{a \in A} w_a.f_a & (1.5) \\ (1 - y_i).(\sum_{a \in \delta^+(i)} f_a - \sum_{a \in \delta^-(i)} f_a - D_i) = 0, \qquad \forall i \in V & (1.6) \\ f_a \geq 0, y_i \in \{0, 1\} \qquad \forall a \in A, \forall i \in V & (1.7) \\ & (1.8) \end{cases}$$

Ce modèle minimise la somme des coûts de transmission (linéaires) et des coûts d'installation de dépôts (fixes) (1.5). Les contraintes (1.6) décrivent les lois de conservation de flots. Celles-ci sont actives uniquement s'il n'y pas de dépôt au nœud considéré ($y_i = 0$). Sinon, le dépôt peut émettre des marchandises sans aucune limite.

Des heuristiques pour résoudre le problème d'UFLP ont été notamment élaborées pour résoudre ce problème NP-difficile, par exemple [64], qui utilisent un algorithme de "passing mes-

sage". D'autre part, [20] propose des considérations probabilistes pour résoudre efficacementent ce problème. On retrouve parmi les méthodes exactes, les travaux de [35] qui propose une approche primal-duale efficace tandis que [44] propose une variante de "Branch & Bound" (appelée "Branch & Peg") dédiée à la résolution de ce problème, qui inclut notamment une heuristique de branchement dynamique. Un grand nombre de variantes existent pour ce problème, comme celui qui y ajoutent des contraintes de capacités sur les liens ("Capacitated Facility Location Problem"), ou plusieurs niveau de dépôts ("$k$-level facility problem).

L'ouvrage [45] rend état de la littérature concernant les problèmes de localisation, selon le point de vue des télécommunications. Qui et al. ([87]) rappellent de leur coté différents types de problèmes de placement dans des réseaux de distribution de contenus, optant pour l'hypothèse majeure que le coût de la mémoire est négligeable par rapport au coût global du serveur. Dans manière général, [87] proposent des heuristiques de placement et les comparent sur des instances réalistes d'internet.

D'autres investigations ont été menées plus précisément sur la localisation des contenus sur les serveurs (et non la localisation des serveurs), de manière à minimiser une fonction de coût sous d'éventuelles contraintes de qualité de service. Les auteurs de [101], par exemple, modélisent le problème de placement de contenus sur un CDN déjà déployé (dont la localisation des serveurs est déjà connue), et se focalisent sur l'optimisation des interactions que peuvent avoir les serveurs entre eux. Ils montrent alors que ce problème, appelé "Replica Placement and Requests Distribution Problem", peut être également vu comme un "Multi-Commodity Capacitated Facility Location Problem".

Ces approches ont en commun de considérer les serveurs d'appoint comme des éléments sous la supervision d'un serveur central, ou d'une intelligence qui en gère le comportement. Une autre manière de construire un CDN est de décentraliser cette intelligence au sein même des serveurs de réplicats, faisant d'eux des dispositifs autonomes capables de réagir au trafic qu'ils constatent. Cela induit une nouvelle contrainte qui crée un nouveau problème de localisation que nous étudierons dans le chapitre 3.

Nous allons présenter les caractéristiques de ces dispositifs autonome : les caches transparents.

## 1.2 Le cache transparent

Le cache, tout comme le serveur de réplicats, a pour fonction de répondre au maximum de requêtes clients qu'il est capable de satisfaire. Il est donc naturellement doté des mêmes équipements que le serveur ; à savoir un (ou plusieurs) espace(s) mémoire(s), et une (ou plusieurs) unité(s) de calcul. Comme son homologue, cet espace peut limiter le volume de fichiers qu'il peut contenir, tandis que le processeur contraint le nombre de requêtes pouvant être simultanément satisfaites. L'intérêt principal du cache est son autonomie, plus particulièrement sa capacité à modifier dynamiquement son contenu dans le but de rester continuellement pertinent au cours du temps, et donc, de maximiser l'interception des requêtes qu'il voit passer à son niveau du réseau. Le mécanisme de mise à jour du cache consiste à reconnaître une requête qui est formulée au nœud (ou au lien) de réseau sur lequel il se trouve, vers le serveur central. Lorsque le contenu, sujet de cette requête, transitera depuis le serveur central jusqu'à l'émetteur de la requête, il rencontrera le cache qui en fera une copie. Ce principe est représenté par la figure 1.4. Ce dispositif agissant

de manière autonome et ne dépendant pas du reste du réseau, on lui donnera souvent le nom plus spécifique de "cache transparent".

**Définition 2.** *Un cache transparent dans un réseau de distribution de contenus est un dispositif électronique et informatique constitué et caractérisé par :*

1. *Une capacité de reconnaissance des requêtes liées à leur dénomination DNS, et une capacité d'interception et de fourniture de ce contenu*
2. *Une capacité de stockage informatique (en octets)*
3. *Une capacité de traitement —capacité de streaming— (généralement exprimée en nombre de sessions simultanées, ou en débit d'octets par seconde)*
4. *Une politique de remplacement des contenus, qui met à jour les fichiers de son espace de stockage*

L'efficacité d'un cache, est généralement mesurée par sa capacité à intercepter des requêtes. C'est-à-dire la probabilité qu'il puisse répondre à une requête. Si l'on ne tient pas compte de la limitation de l'unité de calcul, elle sera directement liée à la capacité de stockage du dit cache. Cette probabilité de succès s'appelle le "Hit Ratio", tandis que son complémentaire s'appelle le "Miss Ratio".

Le Hit ratio se calcule de la manière suivante (Définition 3).

**Définition 3.** *Soit une période $t$, et une librairie de $N_t$ éléments susceptibles d'être demandés sur cette période. Soit un cache et son paramètre $x_{n,t} \in \{0,1\}$ indiquant 1 s'il contient le fichier $n$ sur la période $t$, 0 sinon. Si l'on appelle $D_{n,t}$ le nombre de requêtes qui lui est formulé pour chaque fichier $n$ sur toute la période $t$, le cache aura pour hit ratio sur la période $t$ $HR(t)$ :*

$$HR(t) = \frac{\sum\limits_{n \in N_t} x_{n,t}.D_{n,t}}{\sum\limits_{n \in N_t} D_{n,t}}$$

La performance d'un cache sur la période $t$, mesurée par le hit ratio, va donc uniquement dépendre de sa faculté à posséder le sous-ensemble de fichiers (décrit par $x_{n,t}, \forall n \in N_T$) et qui maximise $\sum\limits_{n \in N_t} x_{n,t}.D_{n,t}$ : c'est-à-dire sa faculté à intercepter les fichiers dont les demandes $D_{n,t}$ sont les plus élevées sur cette période $T$. Si l'on connaissait à l'avance ces demandes, ce problème de choix de contenus deviendrait facile car déterministe. Dans la réalité, ce trafic n'est pas précisément prévisible, cette décision est donc délicate. D'autres indicateurs peuvent également être utilisés pour déterminer la performance d'un cache, notamment calculer non pas le nombre de requêtes satisfaites, mais plutôt le volume de données interceptés.

Pour rester efficace au cours du temps, le cache va donc tendre à contenir les fichiers les plus populaires en chaque instant, et parce que sa taille mémoire est limitée, devoir enlever un ou plusieurs contenus pour pouvoir en ajouter de nouveaux. Cet évènement arrive dès lors que le cache n'a pas été en mesure de satisfaire la requête en cours. Les algorithmes de choix des contenus à supprimer s'appelle les politiques de remplacement. La littérature distingue trois principales politiques de remplacement, LFU ("Least Frequently Used") qui consiste à enlever le fichier en mémoire dont la fréquence de requête est la plus faible, LRU ("Least Recently Used") qui consiste à enlever le fichier en mémoire le plus vieux et enfin RANDOM, qui consiste à retirer un fichier en mémoire au hasard (voir figure 1.5). Bien d'autres politiques existent. Si l'objectif visé de ces politiques est toujours le même, c'est-à-dire maintenir en mémoire les fichiers les plus

populaires du moment, leurs efficacités respectives a fait l'objet de nombreuses études aussi bien expérimentales qu'analytiques ([86], [56], [43], [39]). Il est par exemple admis que "LRU" semble être la politique la mieux à même de capter les corrélations temporelles des requêtes clients [105].

Pour comparer les performances des ces algorithmes, [86], [56], [43], [39] et [105] utilisent généralement un algorithme de stratégie optimale ([91]), connaissant les futures requêtes ([11]), qui va enlever le fichier dont la requête future est la plus éloignée dans le temps. Il est évident que cet algorithme n'est pas, dans la réalité, possible à implémenter puisqu'il suppose une connaissance parfaite des requêtes futures.

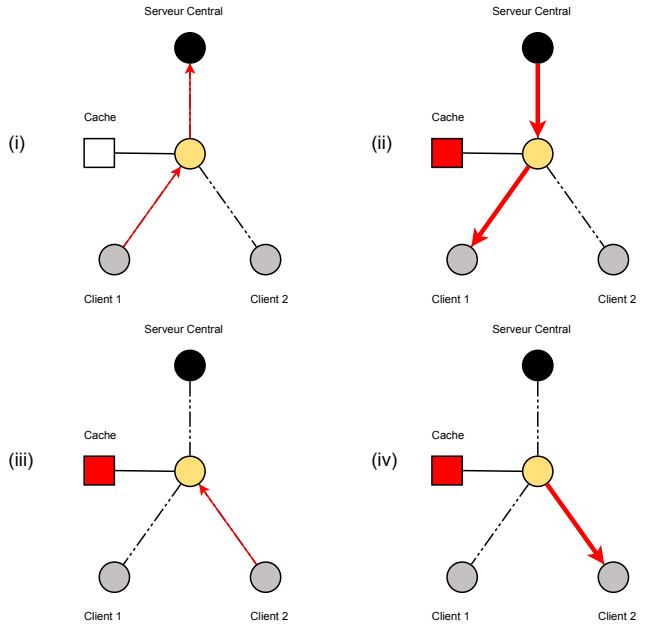



FIGURE 1.4 – *(Le client 1 émet une requête (i), le cache ne peut la satisfaire : elle remonte au serveur central. Le contenu est fourni au client 1 et est stocké dans le cache (ii). Le client 2 émet une requête : elle est interceptée par le cache (iii). Le cache fournit le contenu au client 2 (iv).*

L'intérêt d'un cache est triple. D'une part, puisqu'il est parfaitement autonome, il ne nécessite pas la modification du réseau existant ; il peut se déployer directement sur des liens existants et il est tout à fait sans effet sur le fonctionnement du réseau général. Cela signifie que son coût d'installation reste relativement faible. D'autre part, il permet une plus grande robustesse du réseau face aux pannes. En effet, la mise hors service d'un lien ou d'un routeur ne signifie pas nécessairement qu'une partie des clients n'ont plus accès à un contenu, et ce, même si le serveur central n'est plus accessible. Enfin, et non des moindres, le cache augmente les performances du réseau en terme de rapidité, de qualité de service et limite les effets de congestion qui peuvent survenir dans les moments où le trafic est le plus intense.

Nous avons identifié notre problème d'optimisation comme un problème de localisation. Puisque nous ne voulons pas remettre en cause la structure même du réseau, nous nous plaçons dans le cadre d'un routage des requêtes qui ne sera pas modifié. Plus précisément, dans notre cas le serveur de contenu est le centre du réseau, et c'est vers celui-ci que toutes les de-

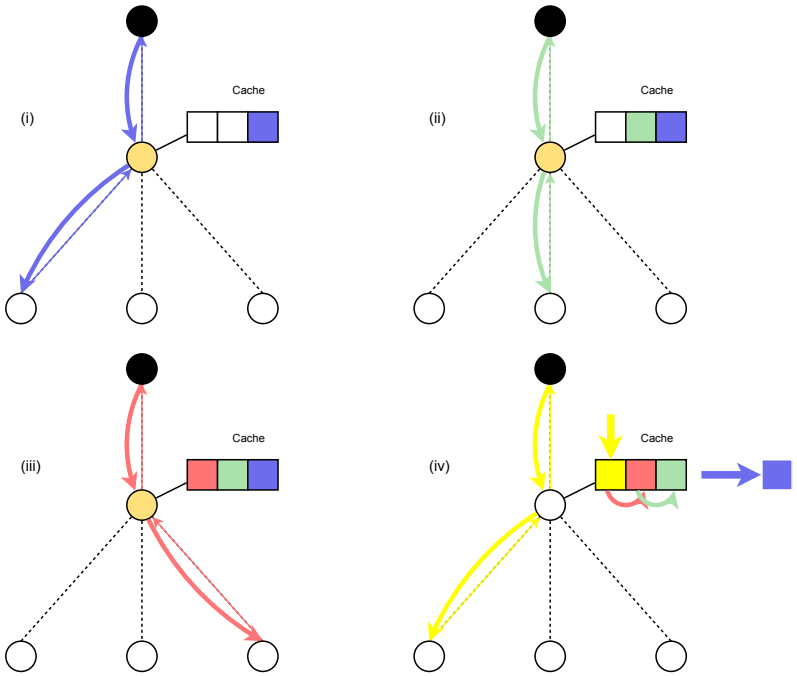



FIGURE 1.5 – *(Plusieurs requêtes sont émises et leurs contenus respectifs sont stockés dans la mémoire du cache (i), (ii) et (iii). La dernière requête et le contenu objet de cette requête nécessitent de retirer un fichier de la mémoire (iv) (ici, le plus ancien, ce qui correspondrait à la politique LRU).*

mandes clients seront acheminées. Le réseau de distribution de contenus induit par les requêtes forme alors une arborescence avec pour racine le serveur central. Notre graphe aura donc la même structure. Par beaucoup d'aspects ce problème de localisation de caches transparents dans les arborescences s'apparente à ce qui est traité dans la littérature.

Si nous nous penchons sur ce qui est fait plus particulièrement dans le cadre des arborescences, on peut noter les quelques références suivantes. Qui et al. ([87]) montrent que dans le cas d'un réseau arborescent si la capacité de distribution du serveur n'est pas limitante, le problème peut être résolu à l'optimum à l'aide d'un programme dynamique polynomial (en $O(N^3M^2)$). Ce programme est tiré de ce qui avait été initialement créé pour la localisation de proxy Web par [68]. S'inspirant de [98] et [87], les auteurs de [61] proposent également une approche de programmation dynamique polynomiale pour un problème connexe. Enfin H. Luss ([72]), propose d'étendre ces mécanismes pour prendre en compte les contraintes particulières de son problème.

Cependant, notre dispositif à déployer possède une stratégie de choix de contenus soumis au trafic lui-même, et plus précisément à la mesure de la popularité qui est faite à son emplacement. Cette spécificité que possède notre problème en plus d'une contrainte de capacité n'a pas été, à notre connaissance, appréhendée dans la littérature pour n'importe quel type de graphe.

D'après ce que nous avons pu observer, la littérature semble suggérer la pertinence des approches de programmation dynamique dans un problème de localisation lorsque le graphe est

une arborescence. On peut donc envisager de construire un algorithme d'optimisation de notre problème en utilisant ce type de paradigme et qui intègre cette contrainte. Pour ce faire nous allons étudier ce qui a déjà été proposé dans la littérature et plus particulièrement deux problèmes de localisation dans des arborescences déjà citées : [61] et [72]. Nous allons détailler ci-après le fonctionnement de chacun de ceux-ci, pour bien comprendre leurs différentes mécaniques et leurs limites. Ceci nous permettra de voir comment nous pouvons nous en inspirer pour déterminer une nouvelle approche générale de programmation dynamique capable de répondre efficacement à notre problématique dans le chapitre 3.

## 1.3 Localisation de caches de Hit Ratio constant dans des graphes arborescents.

Pour les mêmes raisons que [68] et [98], les auteurs de [61] formulent leur problème de localisation optimale de caches en ajoutant deux hypothèses. La première s'appuie sur le routage du trafic Internet qui est effectué selon des plus courts chemins et si l'on considère un serveur (ou grappe de serveurs) central, les chemins de transmission de données formeront une arborescence, ayant pour racine le serveur central. La seconde est qu'ils considèrent le hit ratio $h$ fixé. Les auteurs présentent un problème où il s'agit de déployer un nombre fixé $k$ de caches dans l'arborescence en minimisant la fonction de coût composée du prix de la transmission.

Lorsqu'un cache est placé sur le chemin de routage d'un ou plusieurs flots, celui-ci va alors pouvoir intercepter une partie de ce flot, correspondant au Hit Ratio de ce cache. La figure 1.6 résume l'effet d'un cache au hit ratio connu. Le cache placé au nœud intermédiaire absorbe une partie des requêtes des deux clients placés en aval. Ceci induira une diminution du flot en amont, correspondant à l'efficacité du cache : soit, le Hit Ratio $h$. De ce fait la valeur du trafic (flots) qui sera supporté par l'arc $(oc)$ correspondra à $f = (1-h).(f(1)+f(2))$.

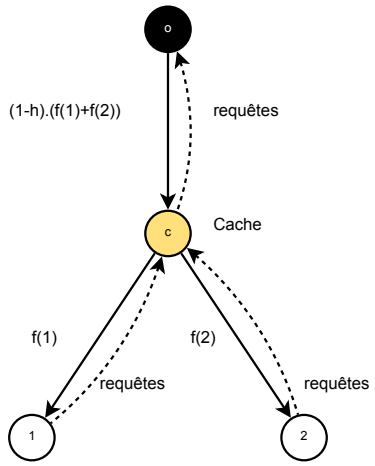



FIGURE 1.6 – *Réseau avec serveur central (en noir), les clients (en blanc) et le cache au nœud intermédiaire.*

Comme nous l'avons déjà mentionné le hit ratio est une mesure de l'efficacité extrêmement dépendante des paramètres suivants ; capacités du dispositif, du trafic, de la politique de remplacement et sa localisation. Ici le hit ratio est prédit comme fixe. C'est à dire que

$HR(t) = \frac{\sum_{n \in N_t} x_{n,t}.D_{n,t}}{\sum_{n \in N_t} D_{n,t}} = h$ est constant quel que soit le moment $(t)$ ou l'emplacement du cache. En réalité, derrière cette simplification se cache l'hypothèse que les caches contiennent l'échantillon de fichiers qui lui permet d'atteindre ce ratio, quel que soit le volume demandé et son origine. Il y a donc de façon induite, une hypothèse de distribution statique de requêtes entre les clients, uniquement pondérée par un volume global. Or cette distribution s'en trouvera nécessairement altérée en amont d'un cache (puisqu'une partie des requêtes auront disparu). Cela signifie que le flot de requêtes d'un client ne pourra être intercepté **qu'une seule fois** par le cache : le reste étant nécessairement des requêtes d'une autre nature.

Le schéma de la figure 1.7 montre un exemple de l'impact des caches sur la distribution de popularité en amont d'un cache. Le cache placé au nœud 2 intercepte les requêtes de premier type (couleur verte) des deux clients. L'interception des requêtes de ce contenu correspond à un hit ratio de 66%. Le second cache en amont (nœud 1) peut avoir dès lors deux hit ratio différents, dépendant du contenu de sa mémoire : à savoir 100% s'il contient le second fichier (couleur rouge) (a), ou 0% s'il contient le premier fichier (couleur verte) (b).

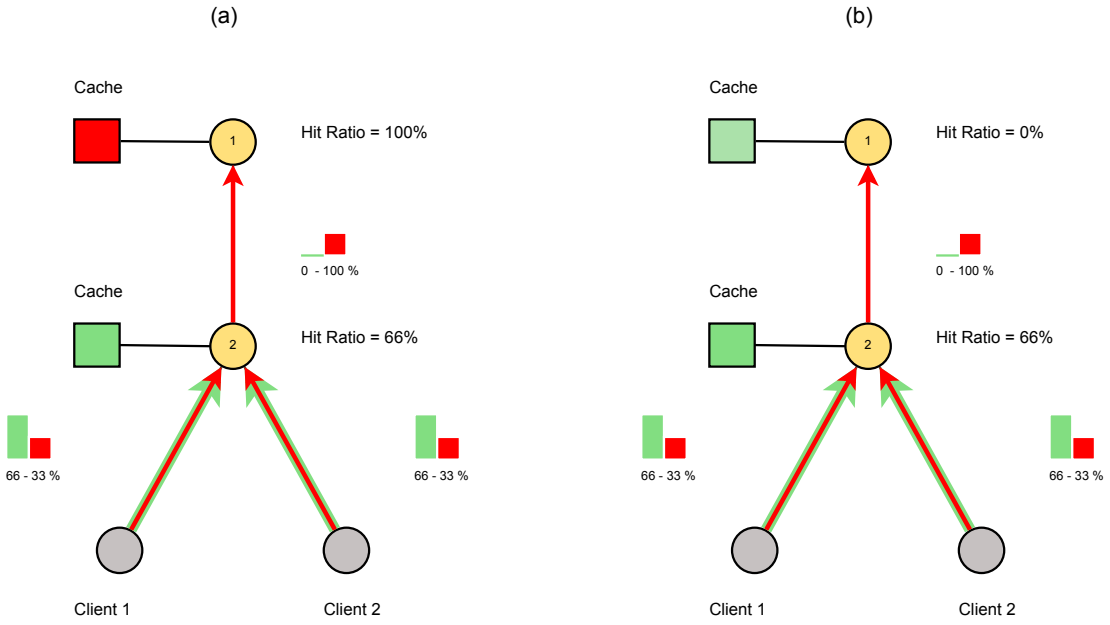



FIGURE 1.7 – *Différents hit ratio, selon le type de contenus présent dans le cache.*

Pour s'affranchir de ce problème, les auteurs de [61] imposent qu'un flot de requêtes d'un client ne soit intercepté que par au plus un cache. Autrement dit, ils segmentent les requêtes entre ce qui est "cachable" et ce qui ne l'est pas. La partie qui ne peut pas être mise en cache sera donc nécessairement routée jusqu'au serveur central. **Cette partie des requêtes est acheminée du serveur central vers les clients quelle que soit la configuration de caches. Le coût induit par leurs transmissions est donc connu et il n'entre pas dans le processus d'optimisation.** Le modèle peut être directement construit sur ce qui est donc "cachable", c'est-à-dire la proportion $h.D_i$ de chaque demande en $i$, comme nous allons le montrer.

On considère une arborescence $\mathcal{T} = (V, A)$ dont la racine $o$ est le serveur central, ainsi que $T \subset V$ des terminaux, ou clients. Chaque client possède sa propre demande, supposée connue, notée $D_i, i \in T$. Le coût de l'arc (unique puisque nous travaillons dans une arborescence) incident à $i$ quand à lui, est noté $w_i, \forall i \in V$. Par convention $w_o = 0$.

Les variables sont divisées en deux types. La mise en place d'un cache est déterminée par la valeur binaire $y_i, \forall i \in V$ de valeur 1 si un cache est placé, 0 sinon. La valeur de transmission de données non "cachables" vers un fils $i$ depuis son père est explicitée par la variable de flots $f_i$. La fonction de coût s'écrit :

$$\sum_{i \in V} f_i.w_i$$

Nous noterons également $\delta^+(i)$ l'ensemble des nœuds fils de $i$. Le problème de localisation de $k$ caches, de hit ratio $h$, dans une arborescence peut s'écrire :

$$k\text{-median} \begin{cases} \min \sum_{i \in V} f_i.w_i, & (1.9) \\ f_i = (1 - y_i).(\sum_{j \in \delta^+(i)} f_j + h.D_i), \qquad \forall i \in V - \{s\}, & (1.10) \\ \sum_{i \in V} y_i \leq k, & (1.11) \\ f_i \geq 0, y_i \in \mathbf{N} \qquad \forall i \in V & (1.12) \end{cases}$$

La fonction de coût (1.9) est la somme des flots de l'arborescence, tandis que la contrainte (1.11) limite le nombre d'installation maximum effectuées. Le flot $f_i$ est la somme des flots de demandes des nœuds fils $\delta^+(i)$ de $i$ plus la demande $h.D_i$ au nœud $i$ (partie pouvant être interceptée par un cache). Si un cache est présent, alors cette demande est satisfaite par ce cache, et tous les flots ascendant sont éliminés, ce qui est représenté par la contrainte (1.10). Si la contrainte (1.10) peut être linéarisée simplement, les auteurs proposent plutôt d'utiliser un algorithme de programmation dynamique dont nous allons voir le fonctionnement.

A chaque nœud de l'arbre le problème est décomposé en un choix entre deux décisions : mettre un cache, ou ne pas en mettre. Il faudra alors prendre la décision qui nous coûte le moins cher entre les deux. Nous définissons $C(i, l, k)$, le coût de la solution optimale des sous-arbres induits par le nœud $i$ avec un maximum de $k$ caches qui peuvent encore être installés et une distance $l$ (au sens des longueurs $w_i$) la plus faible entre $i$ et un cache en amont. Il s'agira alors de trouver la meilleure solution $C^*(i, l, k)$ entre les deux possibilités (mettre un cache ou non), soit :

**Proposition 1.** *Le coût optimal, noté $C^*(i, l, k)$, de la solution du sous-arbre induit (et comprenant) par $i$, sachant le coût de transmission vers le cache en amont le plus proche $l$ (ou distance), et un nombre $k$ de caches disponibles, est donné par la relation :*

$$C^*(i, l, k) = \min\{C(i, l, k) + h.D_i.l, C(i, 0, k - 1)\} \tag{1.13}$$

*Démonstration.* La solution optimale en $i$, connaissant la longueur $l$ et un nombre de caches $k$ disponibles, sera donc celle qui sera de coût le plus faible entre le coût de la solution optimale où le choix est fait de placer un cache en $i$ et le coût de la solution optimale où le choix est fait de ne pas placer de cache $i$, auquel s'additionne un coût de transmission de la demande $D_i$ vers le cache en amont le plus proche ($h.D_i.l$) dans ce dernier cas.

□

Or, la solution optimale va être constituée de la meilleure combinaison de ses sous-arbres fils correspondant, étant donnée la prise de décision en $i$.

**Proposition 2.** *Le coût optimal de la solution des sous-arbres induits par $i$, ayant fixé l'état de $i$, $k$ caches restant disponibles et de coût de transmission vers le cache le plus proche $l$ (ou distance), et noté $C(i,l,k)$, est donné par la relation :*

$$C(i,l,k) = \min_{\substack{k_j \in \mathbf{N}, \forall j \in \delta^+(i) \\ \sum_{j \in \delta^+(i)} k_j = k}} \{ \sum_{j \in \delta^+(i)} C^*(j, l + w_j, k_j) \} \tag{1.14}$$

*Démonstration.* Soit $i$ le nœud courant, et l'arborescence $\mathcal{T}^i - \{i\}$ induite par $i$ et privée de $i$. Toute solution réalisable du problème d'optimisation de localisation de caches généré par une sous-arborescence $T^j, j \in \delta^+(i)$, constitue nécessairement une solution réalisable au problème de localisation sur $\mathcal{T}^i - \{i\}$, si la somme de tous les caches placés sur celle-ci n'excède pas $k$. Or, le problème de minimisation de $C(j,l,k)$, connaissant une configuration de placement en amont a uniquement pour données d'entrée les paramètres suivant :

1. le graphe de la sous-arborescence de racine $j$ (la sous-arborescence elle-même) et ses coûts associés
2. la distance $l$ avec le cache/serveur en amont le plus proche
3. le nombre $k_j$ de caches disponibles pour la sous-arborescence $j$

Si l'on choisi une combinaisons d'allocation $k_j$ pour chaque sous-arborescence $j$, telle que $\sum_{j \in \delta^+(i)} k_j = k$, alors le coût de la solution optimale sur l'arborescence induite par $i$ correspond à la somme des coûts des solutions optimales des sous-arborescences induites par $j \in \delta^+(i)$.
Le coût optimal de la solution de l'arborescence induite par $i$ (privée de $i$), correspond donc au minimum de la somme des coûts des solutions optimales des sous-arborescences, parmi toutes les combinaisons d'allocations de caches $k_j, j \in \delta^+(i)$, telles que $\sum_{j \in \delta^+(i)} k_j = k$. □

La différence fondamentale entre $C(i,l,k)$ et $C^*(i,l,k)$, est que pour $C(i,l,k)$ la décision de placement de cache en $i$ est déjà prise : c'est uniquement le coût de la solution optimale considérant un état déjà défini en $i$, tandis que $C^*(i,l,k)$ détermine la meilleure de ces deux décisions (mettre un cache ou non). Pour connaître la solution optimale de toute l'arborescence, il suffira alors de calculer $C^*(o,0,K)$.

Une manière de contracter les deux relations de récurrence :

$$C^*(i,l,k) = \min \begin{cases} \min\limits_{\substack{k_j \in \mathbf{N}, \forall j \in \delta^+(i) \\ \sum_{j \in \delta^+(i)} k_j = k}} \{ \sum\limits_{j \in \delta^+(i)} C^*(j, l + w_j, k_j) \} \\ \min\limits_{\substack{k_j \in \mathbf{N}, \forall j \in \delta^+(i) \\ \sum_{j \in \delta^+(i)} k_j = k-1}} \{ \sum\limits_{j \in \delta^+(i)} C^*(j, w_j, k_j) \} \end{cases} \tag{1.15}$$

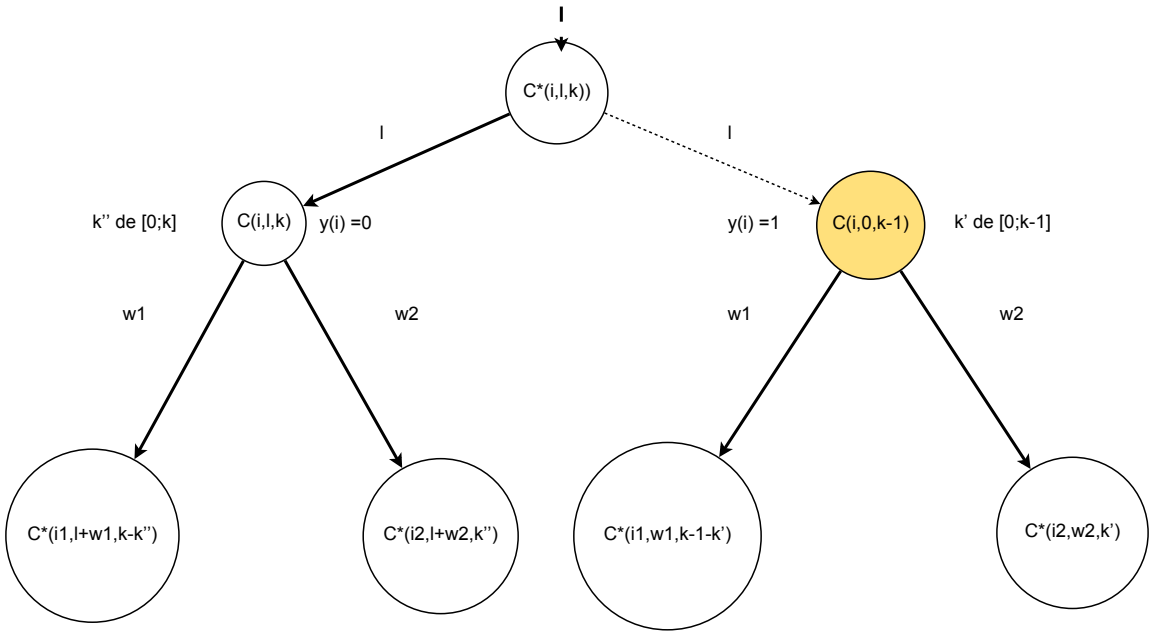



FIGURE 1.8 – *Arbre de décision de placement de caches (partie haute l'arbre 1.13), puis description des sous-problèmes induits par chaque décision (partie basse de l'arbre correspondant à 1.14).*

Nous obtenons une récursion en deux étapes, où le coût optimal $C^*(i,l,k)$ représente la meilleure décision à prendre entre installer un cache ou ne pas en installer. Cette décision va dépendre de la combinaison optimale de partitionnement des $k$ caches disponibles $C(i,l,k)$.

La figure 1.8 résume le principe des relations de récurrence. Afin d'améliorer les performances théoriques et pratiques de l'algorithme, les auteurs effectuent une transformation sur le graphe en ajoutant des nœuds virtuels de sorte à ce qu'un père ait au plus deux fils : $i_l$ et $i_r$. Le pseudo-code de l'algorithme est décrit ci-dessous.

**Algorithm 1** Localisation de caches ($C^*(i,l,k)$)

**Require:** $\mathcal{T} = \mathcal{G}(\mathcal{V}, \mathcal{A})$ arborescence de racine $i$, longueur vers le cache en amont le plus proche $l$, nombre de caches disponibles $k$

**Ensure:** Meilleure solution $C^*(i,l,k)$ entre les deux décisions (mettre ou ne pas mettre de cache)

**return** $\min\{C(i,0,k-1), C(i,l,k)\}$

**Algorithm 2** Localisation de caches ($C(i,l,k)$)

**Require:** $\mathcal{T} = \mathcal{G}(\mathcal{V}, \mathcal{A})$ arborescence de racine $i$, longueur vers le cache en amont le plus proche $l$, nombre de caches disponibles $k$
**Ensure:** Meilleure solution $C(i,l,k)$ induite par l'état $(i,l,k)$
  $C(i,l,k)$ solution initialisée à l'infini.
  **for all** $k' \in \mathbf{N}, 0 \leq k' \leq k$ **do**
    **if** $C(i,l,k) > C^*(i_l, l + w_{i_l}, k') + C^*(i_r, l + w_{i_r}, k - k') + l.D_i.h$ **then**
      $C(i,l,k) \leftarrow C^*(i_l, l + w_{i_l}, k') + C^*(i_r, l + w_{i_r}, k - k') + l.D_i.h$
    **end if**
  **end for**
  **return** $C(i,l,k)$

Cette approche a été prouvée de complexité $O(kn^2)$ par [98].

Un problème très proche peut également être traité de cette manière. A la place d'une contrainte d'un nombre $K$ de caches disponibles à placer (contrainte (1.11) du modèle précédent), nous pouvons attribuer un coût $c_c$ à la mise en place d'un cache. Nous obtiendrons alors le modèle suivant :

$$\text{Localisation de caches} \begin{cases} \min \sum_{i \in V} f_i.w_i + y_c.c_c, & (1.16) \\ f_i = (1 - y_i).(\sum_{j \in \delta^+(i)} f_j + h.D_i), \quad \forall i \in V - \{s\}, & (1.17) \\ f_i \geq 0, y_i \in \mathbf{N} \quad \forall i \in V & (1.18) \end{cases}$$

Ce modèle s'assimile au problème plus classique d'Uncapacitated Facility Location Problem déjà cité, dans le cas particulier d'une arborescence. A nouveau, le problème peut être traité de manière polynomiale par un algorithme de programmation dynamique -de complexité $--O(n)$. Le problème est notamment plus simple, car d'une décision effectuée au père, les sous-solutions correspondant à ses fils deviennent indépendantes entre elles, ce qui n'était pas le cas précédemment (liées entre elles par les $k$ caches disponibles).

**Proposition 3.** *Le coût optimal, noté $C^*(i,l)$, de la solution du sous-arbre induit (et comprenant) par $i$, sachant le coût de transmission vers le cache en amont le plus proche $l$ (distance) est donné par la relation :*

$$C^*(i,l) = \min\{C(i,l) + h.D_i.l, C(i,w_i) + c_c\} \tag{1.19}$$

*Démonstration.* La solution optimale en $i$, connaissant la longueur $l$, sera donc celle qui sera de coût le plus faible entre le coût de la solution optimale où le choix est fait de placer un cache en $i$ et le coût de la solution optimale où le choix est fait de ne pas placer de cache en $i$, auquel s'additionne un coût de transmission de la demande $D_i$ vers le cache en amont le plus proche ($h.D_i.l$) dans ce dernier cas.

□

Or, la solution optimale va être constituée de la meilleure combinaison de ses sous-arbres fils correspondants, étant donnée la prise de décision en $i$.

**Proposition 4.** *Le coût optimal de la solution des sous-arbres induits par $i$, ayant fixé l'état de $i$, un coût de transmission vers le cache le plus proche $l$, et noté $C(i,l)$, est donné par la relation :*

$$C(i,l) = \sum_{j \in \delta^+(i)} C^*(j,l) \tag{1.20}$$

*Démonstration.* Soit $i$ le nœud courant, et l'arborescence $\mathcal{T}^i - \{i\}$ induite par $i$ et privée de $i$. Toute solution réalisable du problème d'optimisation de localisation de caches généré par une sous-arborescence $\mathcal{T}^j, j \in \delta^+(i)$, constitue nécessairement une solution réalisable au problème de localisation sur $\mathcal{T}^i - \{i\}$. Or, le problème de minimisation de $C(j,l)$, connaissant une configuration de placement en amont a uniquement pour données d'entrée les paramètres suivant :

1. le graphe de la sous-arborescence de racine $j$ (la sous-arborescence elle-même) et ses coûts associés
2. la distance $l$ avec le cache/serveur en amont le plus proche
3. le nombre $c_c$ coût d'installation de cache

Le coût optimal de la solution de l'arborescence $\mathcal{T}^i - \{i\}$, correspond donc au minimum de la somme des coûts des solutions optimales des sous-arborescences $\mathcal{T}^j, j \in \delta^+(i)$. □

Ce qui peut être contracté sous la formule unique :

$$C^*(i,l) = \min \begin{cases} \sum\limits_{j \in \delta^+(i)} C^*(j,w_j) + c_c \\ \sum\limits_{j \in \delta^+(i)} C^*(j,l) + h.D_j.l \end{cases} \tag{1.21}$$

Si l'hypothèse d'un graphe arborescent s'explique par des tables de routages fixes, il est en revanche impossible de traiter le cas de caches hiérarchiques ; ce sont là les limites des hypothèses de ce papier pour intégrer le comportement des caches transparents. Pour pouvoir traiter comportement, il nous faudrait alors la connaissance précise des différentes requêtes et des contenus associés à chaque nœud du réseau. C'est ce qui nous motive à présenter le second article qui considère le placement des contenus en sus de la localisation des dispositifs.

## 1.4 Localisation de serveurs et de contenus dans des graphes arborescents

L'un des cas d'usage pour lequel il est nécessaire de déployer un CDN est le service de vidéo à la demande. En effet ce type de service offre un contenu multimédia massif (catalogue comprenant généralement plusieurs milliers de films) et de taille importante (de 1 à 5 Go par film selon la qualité d'encodage) pour un marché extrêmement grand (plusieurs dizaines de millions de transactions estimées par an). C'est ce qui motive H. Luss dans l'étude de la localisation optimale de serveurs et de contenus.

Plus précisément Hanan Luss ([72]), propose de déployer un CDN dans un réseau arborescent $\mathcal{T} = \mathcal{G}(\mathcal{V}, \mathcal{A})$ ayant pour racine le serveur central $o$ pour de la vidéo à la demande (VOD), en spécifiant le contenu associé à chaque serveur : il appelle ce problème le SLPAM (pour "Service Location and Program Assignement Model"). Le catalogue disponible pour la VoD est décrit par l'ensemble de ses classes $p = 1, 2, ..., |P|$, qui représente les $|P|$ catégories de films (comédies, thriller, etc...) disponibles. Les clients, pouvant être localisés à n'importe quel nœud $i$, formulent alors un sous-ensemble $D_i$ de demandes dans ce catalogue $P$. Luss suppose également que le

mode de transmission est le multi-cast : l'information peut être répliquée en chaque nœud lors de sa transmission. De cette manière, la même information qui parcourt un lien donné peut servir plusieurs demandes, comme le représente la figure 1.9. De ce fait, les variables explicitant un flot de section $p$ sera binaire.

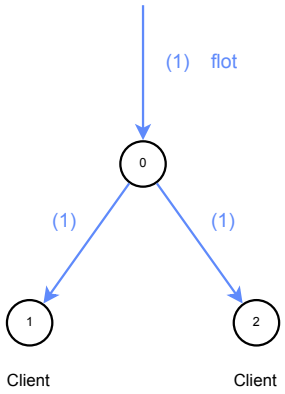



FIGURE 1.9 – *Le flot d'information arrivant sur le nœud 0, est répliqué pour être transmis à la fois au client 1 et au client 2.*

Cela suppose une certaine corrélation temporelle entre les requêtes. De ce fait, la consommation de bande passante (modélisée ici par un flot) n'est qu'une somme d'informations strictement différentes.

Les coûts sont partagés entre trois composants : l'installation de serveurs, de coût $C_s$, la mise en place de classe $p$ au sein d'un serveur, de coût $C_p$ et enfin le coût de transmission $w_{i,p}$ d'une classe de fichier $p$ sur l'arc incident à $i$ ($w_{o,p} = 0, \forall p \in P$). L'auteur introduit alors les ensembles $QN(i,j)$ qui indiquent les nœuds qui constituent le chemin unique de l'arborescence $\mathcal{T}$ reliant $i$ à $j$.

Les variables sont les suivantes :

1. $y_i$ : la variable de décision binaire de valeur 1 si un serveur est installé au nœud $i$, 0 sinon.
2. $x_{i,p}$ : la variable de décision binaire de valeur 1 si la classe de films $p$ est contenue dans le serveur du nœud $i$ ; 0 sinon.
3. $u_{i,j,p}$ : la variable de décision binaire de valeur 1 si le serveur du nœud $i$ est le serveur émetteur du contenu $p$ pour le client $j$, 0 sinon.
4. $b_{i,p}$ : la variable de décision binaire de valeur 1 si le contenu $p$ est transmis sur le lien incident à $i$.

[72] formule alors le modèle suivant :

$$
\text{SLPAM}\begin{cases}
\min \sum_{i \in V} C_s.y_i + (\sum_{p \in P} C_p.x_{i,p} + w_{i,p}.b_{i,p}) & & (1.22)\\
\sum_{i \in QN(o,j)} u_{i,j,p} = 1, & \forall p \in D_j, j \in V & (1.23)\\
x_{i,p} \leq y_i, & \forall p \in P, j \in V & (1.24)\\
u_{i,j,p} \leq x_{i,p}, & \forall p \in D_j, i \in QN(0,j), j \in V & (1.25)\\
b_{j,p} \geq u_{i,k,p}, & \forall p \in D_j, j \in QN(i,k) - \{i\} \; k \in V & (1.26)\\
y_0 = 1, y_i \in \{0,1\}, \forall i \in V & & (1.27)\\
x_{o,p} = 1, x_{i,p} \in \{0,1\}, \forall i \in V & & (1.28)\\
u_{i,j,p} \in \{0,1\}, \forall j \in V, p \in D_j \; i \in QN(o,j) & & (1.29)\\
b_{i,p} \in \{0,1\}, \forall i \in V, p \in P & & (1.30)
\end{cases}
$$

Le modèle explicite une fonction de coût (1.22) reprenant toutes les sources de coût du problème, soit : l'installation des serveurs, des contenus en ceux-ci et la transmission sur les liens. La contrainte (1.23) impose qu'à chaque demande client, un serveur situé sur le chemin du serveur central au client satisfasse cette demande. Les contraintes (1.24) et (1.25), quant à elles, sont des contraintes classiques imposant qu'un serveur, pour être émetteur d'un contenu pour un —ou plusieurs— clients, doit avoir un serveur et que ce serveur contienne le dit contenu. La dernière contrainte (1.26), impose enfin que les liens du chemin reliant le serveur émetteur d'un contenu pour un client jusquà celui-ci, transmettent effectivement ce contenu .

Puisque le problème n'est constitué que de variables binaires, l'auteur suggère une approche de programmation dynamique. Pour cela, il explicite trois états différents que peut avoir chaque classe de films de chaque nœud.

**Définition 4.** *Pour tout programme $p$ et tout nœud $i$ nous appelons états $e_{ip}$ les états selon lesquels :*

1. $e_{ip} = 0$ *: signifie que le programme p ne transite pas par le nœud i. Cela signifie notamment qu'il ne peut en être l'émetteur ($x_{ip} = 0$) mais également que $b_{ip} = 0$).*
2. $e_{ip} = 1$ *: signifie que le programme est émis de ce nœud, ce qui implique à la fois qu'il y a un serveur $y_i = 1$ et que le programme est contenu dans celui-ci $x_{ip} = 1$.*
3. $e_{ip} = 2$ *: signifie que le programme n'est pas installé sur le nœud i, mais transite sur son lien incident, soit que $b_{ip} = 1$.*

L'intérêt de l'ensemble de ces états est qu'il décrit de manière exhaustive les effets des toutes les configurations possibles en amont de ce nœud, permettant de décomposer le problème ce qui permet l'utilisation de la programmation dynamique.

Le vecteur $e_i = (e_{i1}, e_{i2}, ..., e_{i|P|})$ désignera alors le vecteur d'état total du nœud $i$. De plus, la solution optimale est caractérisée par la propriété que chaque demande client d'un contenu est délivrée par le serveur doté de ce contenu le plus proche de ce client (du fait de la minimisation du coût de la transmission). Dès lors, nous pourrons alors décrire la solution optimale uniquement par un vecteur optimal de composantes $e_i^*, \forall i \in V$. Nous réutiliserons également la notation $y_i(e_i)$, de valeur 1, si il existe au moins un $e_{ip} = 1, \forall p \in P$, qui correspond à la nécessité

de placer un serveur à cet endroit.

L'auteur introduit alors le coût local d'un nœud d'après son état, soit :

$$C(e_i) = y_i(e_i).C_s + \sum_{\substack{e_{ip}=1 \\ p \in P}} C_p + \sum_{\substack{e_{ip}=2 \\ p \in P}} w_{i,p} \tag{1.31}$$

Il distingue ce dernier du coût de la solution optimale locale, lié au sous-arbre induit par $i$ (arborescence $\tau_i$ de racine $i$) par $C(\tau_i, e_i)$, après avoir choisi l'état $e_i$. Luss construit également l'ensemble suivant :

**Définition 5.** *L'ensemble $REAL_i(e_j)$ désigne tous les états possibles $e_i$ de $i$ sachant $e_j$, avec $j = \delta^-(i)$ (nœud d'origine de l'arc unique incident à $i$).*

La solution locale optimale est construite parmi ces différents états : $C^*(\mathcal{T}^i, e_j)$, en s'assurant que l'état $e_i^*$ est réalisable pour $e_j$. C'est à dire que la solution $e_i$ associée en $i$ fait partie de $REAL_i(e_j)$.

**Proposition 5.** *Le coût de la solution optimale de la sous-arborescence $\mathcal{T}^i$ sachant $e_j$, , avec $j = \delta^-(i)$ (nœud d'origine de l'arc unique incident à $i$) est noté $C^*(\mathcal{T}^i, e_j)$, et donné par la relation :*

$$C^*(\mathcal{T}^i, e_j) = \min_{e_i \in REAL_i(e_j)} C(\mathcal{T}^i, e_i) \tag{1.32}$$

*Démonstration.* Si la solution de la sous-arborescence induite par $i$ est réalisable pour $j$ et son état déterminé $e_j$, alors elle est réalisable pour toute l'arborescence (récursion de la garantie de réalisabilité). Le coût de la solution optimale de la sous-arborescence induite par $i$ sachant l'état $e_j$ du nœud père, est donc le minimum des coûts des solutions des états $e_i$ au nœud $i$ réalisables sachant $e_j$. □

Or, le coût de l'état de la solution optimale locale sachant l'état $e_i$ fixé s'obtient de la manière suivante :

**Proposition 6.** *Le coût de la solution optimale de la sous-arborescence $\mathcal{T}^i$ muni d'un état $e_i$ fixé, noté $C(\mathcal{T}^i, e_i)$, est donné par la relation :*

$$C(\mathcal{T}^i, e_i) = C(e_i) + \sum_{j \in \delta^+(i)} (C^*(\tau_j, e_i)) \tag{1.33}$$

*Démonstration.* Le coût de la solution optimale de la sous-arborescence induite par $i$ sachant l'état $e_i$ au nœud $i$, est donné par la somme des coûts locaux (définis par $C(e_i)$) et de la somme des solutions optimales des sous-arborescences induites par $i$. □

Si $C^*(\mathcal{T}^i, e_j)$ exprime le coût de la solution optimale sous contrainte de l'état $e_j$ donné par le nœud origine de son arc incident, $C(\mathcal{T}^i, e_i)$ exprime le coût de la solution optimale induit par le choix de l'état $e_i$ sur la racine $i$ du sous-arbre $\mathcal{T}^i$. La relation (1.32), va nous permettre alors de les lier, en s'assurant la cohésion des états $e_i$ et $e_j$ par l'ensemble de réalisabilité $REAL_i(e_j)$, et ce, de proche en proche.

Comme précédemment, les relations de récurrences sont représentées dans le schéma de la figure 1.10.

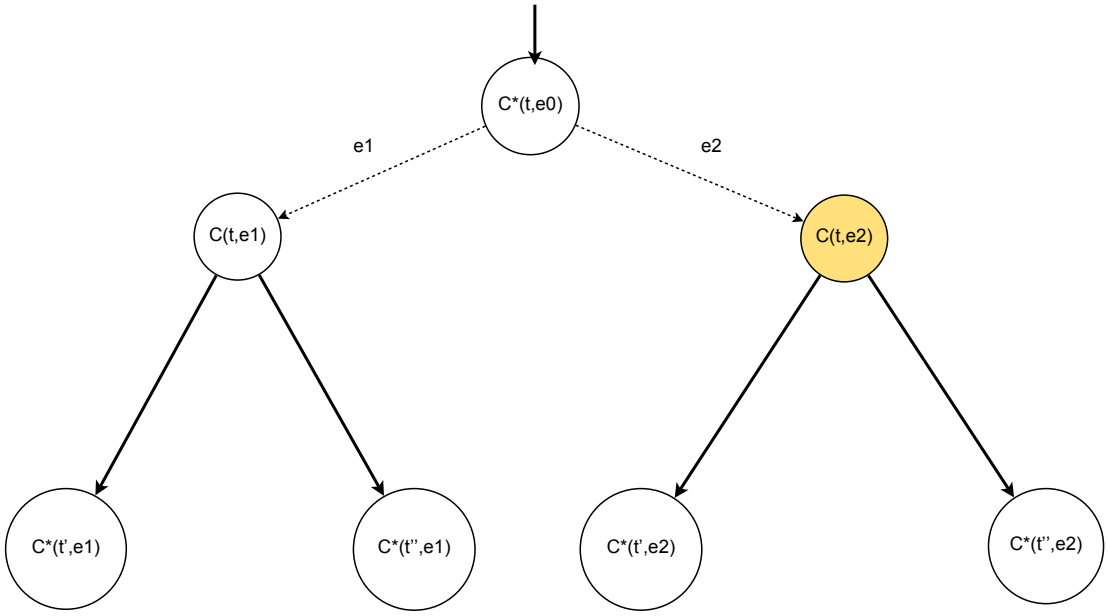



FIGURE 1.10 – *Arbre de décision d'états (partie haute l'arbre (1.32)), puis description des sous-problèmes dans l'arbre réel induits par chaque décision d'états (partie basse de l'arbre correspondant à (1.33)).*

Enfin l'algorithme peut se présenter sous la forme suivante :

**Algorithm 3** Localisation de serveurs et de contenus $C^*(\mathcal{T}^i, e_k)$

**Require:** $\mathcal{T} = \mathcal{G}(\mathcal{V}, \mathcal{A})$ arborescence de racine $i$, $P$ ensemble des programmes, $e_k$ état courant.
**Ensure:** Meilleure solution $C^*(\mathcal{T}^i, e_k)$

$REAL_i$, ensembles des états réalisables, pour chaque $i \in V$
**for all** $e_i \in REAL_i(e_k)$ **do**
  **if** $\delta^+(i) \neq \emptyset$ **then**
    **for all** $j \in \delta^+(i)$ **do**
      Calculer et enregistrer $C^*(\tau_j, e_i)$ (1.32)
    **end for**
  **end if**
  Calculer $C(\mathcal{T}^i, e_i)$ (1.33)
**end for**
**return** $C^*(\mathcal{T}^i, e_k) = \min_{e_i \in REAL_i(e_k)} \{C(\mathcal{T}^i, e_i)\}$

Nous allons à présent montrer un exemple du fonctionnement de l'algorithme. Nous utilisons l'instance de la figure 1.11, et de trois classes de contenus $p$. Partant de l'état initial du serveur central $(1, 1, 1)$, nous chercherons alors l'état optimal $C^*(\tau_1, (1, 1, 1))$ du seul nœud fils 1. De façon récursive nous serons donc amenés à calculer les solutions optimales alternativement $C^*$ et $C$ jusqu'à atteindre les feuilles de l'arbre. L'auteur propose de sauvegarder les états déjà calculés afin de minimiser l'effort de calcul. Les tableaux 1.1, 1.2, 1.3 et 1.4 réfèrent les états des nœuds potentiellement calculés, et la manière dont ils sont calculés. Lorsque l'algorithme stoppe,

nous obtenons alors la solution optimale décrite sur la figure 1.12, et qui nous est donné par "backtracking" des états optimaux comme nous le présente le tableau 1.5.

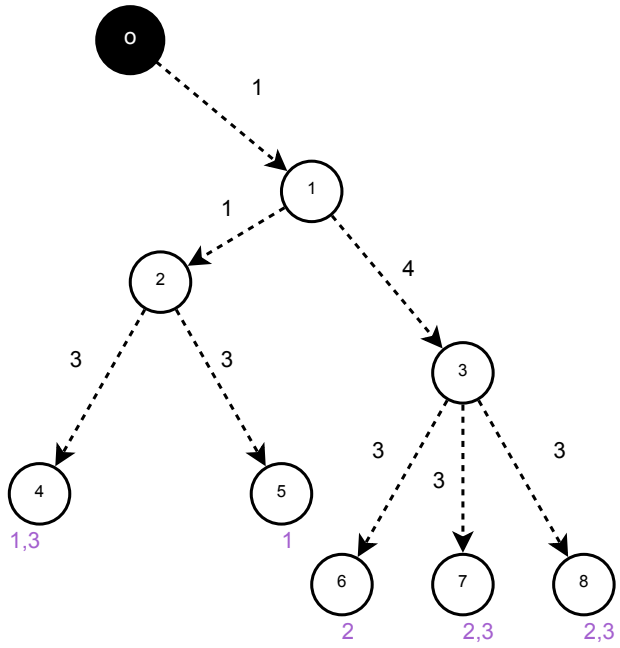



FIGURE 1.11 – *Arborescence de racine o. Les coûts de transmission sur les liens sont indépendants des contenus qui y transitent et sont notés en noir. Les classes de contenus sont définis par* $P = \{1, 2, 3\}$. *Les demandes en chaque nœud sont notées en violet. Enfin le coût d'un serveur est* $C_s = 6$ *et le coût d'installation de contenus* $C_1 = C_2 = C_3 = 2$.

| Nœud 4 | | Nœud 5 | |
|---|---|---|---|
| $REAL_4(e_2)$ | $C(\tau_4, e_4)$ | $REAL_5(e_2)$ | $C(\tau_5, e_5)$ |
| (1,0,1) | 6+2+2=10 | (1,0,0) | 6+2=8 |
| (1,0,2) | 6+2+3=11 | (2,0,0) | 3 |
| (2,0,1) | 6+3+2=11 | | |
| (2,0,2) | 3+3=6 | | |

| Nœud 6 | | Nœud 7/8 | |
|---|---|---|---|
| $REAL_6(e_3)$ | $C(\tau_6, e_6)$ | $REAL_{7/8}(e_3)$ | $C(\tau 7/8, e_{7/8})$ |
| (0,1,0) | 6+2=8 | (0,1,1) | 6+2+2=10 |
| (0,2,0) | 3 | (0,1,2) | 6+2+3=11 |
| | | (0,2,1) | 6+3+2=11 |
| | | (0,2,2) | 3+3=6 |

TABLE 1.1 – Les états possibles des nœuds 4 à 8.

| $REAL_3(e_1)$ | $C(\tau_3, e_3)$ |
|---|---|
| (0,0,0) | $0 + C(\tau_6,(0,1,0)) + C(\tau_7,(0,1,1)) + C(\tau_8,(0,1,1)) = 8+10+10 = 28$ |
| (0,0,1) | $6 + 2 + C(\tau_6,(0,1,0)) + C(\tau_7,(0,1,1)) + C(\tau_8,(0,1,1)) = 8+8+10+10 = 36$ |
| (0,0,2) | $4 + C(\tau_6,(0,1,0)) + C(\tau_7,(0,1,1)) + C(\tau_8,(0,1,1)) = 4+8+10+10 = 32$ |
| (0,1,0) | $6 + 2 + C(\tau_6,(0,2,0)) + C(\tau_7,(0,1,1)) + C(\tau_8,(0,1,1)) = 8+3+10+10 = 31$ |
| (0,1,1) | $6 + 2 + 2 + C(\tau_6,(0,2,0)) + C(\tau_7,(0,2,2)) + C(\tau_8,(0,2,2)) = 10+3+6+6 = 25$ |
| (0,1,2) | $6 + 2 + 4 + C(\tau_6,(0,2,0)) + C(\tau_7,(0,2,2)) + C(\tau_8,(0,2,2)) = 12+3+6+6 = 27$ |
| (0,2,0) | $4 + C(\tau_6,(0,2,0)) + C(\tau_7,(0,1,1)) + C(\tau_8,(0,1,1)) = 4+3+10+10 = 27$ |
| (0,2,1) | $6 + 2 + 4 + C(\tau_6,(0,2,0)) + C(\tau_7,(0,2,2)) + C(\tau_8,(0,2,2)) = 12+3+6+6 = 27$ |
| (0,2,2) | $4 + 4 + C(\tau_6,(0,2,0)) + C(\tau_7,(0,2,2)) + C(\tau_8,(0,2,2)) = 8+3+6+6 = 23$ |

| $REAL_3(e_1)$ | $C(\tau_3, e_3)$ |
|---|---|
| (0,0,0) | $8+10+10 = 28$ |
| (0,0,1) | $8+8+10+10 = 36$ |
| (0,0,2) | $4+8+10+10 = 32$ |
| (0,1,0) | $8+3+10+10 = 31$ |
| (0,1,1) | $10+3+6+6 = 25$ |
| (0,1,2) | $12+3+6+6 = 27$ |
| (0,2,0) | $4+3+10+10 = 27$ |
| (0,2,1) | $12+3+6+6 = 27$ |
| (0,2,2) | $8+3+6+6 = 23$ |

TABLE 1.2 – Les états possibles du nœud 3. La solution optimale, sans état contraignant, serait donc $C^*(\tau_3, -) = 23$.

| $REAL_2(e_1)$ | $C(\tau_2, e_2)$ |
|---|---|
| (0,0,0) | $0 + C(\tau_4,(1,0,1)) + C(\tau_5,(1,0,0)) = 18$ |
| (0,0,1) | $6 + 2 + C(\tau_4,(1,0,1)) + C(\tau_5,(1,0,0)) = 26$ |
| (0,0,2) | $1 + C(\tau_4,(1,0,1)) + C(\tau_5,(1,0,0)) = 19$ |
| (1,0,0) | $6 + 2 + C(\tau_4,(1,0,1)) + C(\tau_5,(2,0,0)) = 21$ |
| (1,0,1) | $6 + 2 + 2 + C(\tau_4,(2,0,2)) + C(\tau_5,(2,0,0)) = 19$ |
| (1,0,2) | $6 + 2 + 1 + C(\tau_4,(2,0,2)) + C(\tau_5,(2,0,0)) = 18$ |
| (2,0,0) | $1 + C(\tau_4,(1,0,1)) + C(\tau_5,(2,0,0)) = 14$ |
| (2,0,1) | $6 + 2 + 1 + C(\tau_4,(2,0,2)) + C(\tau_5,(2,0,0)) = 18$ |
| (2,0,2) | $1 + 1 + C(\tau_4,(2,0,2)) + C(\tau_5,(2,0,0)) = 11$ |

TABLE 1.3 – Les états possibles du nœud 2.

| $REAL_1(e_o)$ | $C(\tau_1, e_2)$ |
|---|---|
| (0,0,0) | $0 + C(\tau_2,(0,0,0)) + C(\tau_3,(0,1,1)) = 18 + 25 = 43$ |
| (0,0,1) | $6 + 2 + C(\tau_2,(0,0,0)) + C(\tau_3,(0,1,1)) = 8 + 18 + 25 = 51$ |
| (0,0,2) | $8 + C(\tau_2,(0,0,0)) + C(\tau_3,(0,1,1)) = 8 + 18 + 25 = 51$ |
| (0,1,0) | $6 + 2 + C(\tau_2,(0,0,0)) + C(\tau_3,(0,1,1)) = 8 + 18 + 25 = 51$ |
| (0,1,1) | $6 + 2 + 2 + C(\tau_2,(0,0,0)) + C(\tau_3,(0,2,2)) = 10 + 18 + 23 = 51$ |
| (0,1,2) | $6 + 2 + 8 + C(\tau_2,(0,0,0)) + C(\tau_3,(0,2,2)) = 16 + 18 + 23 = 57$ |
| (0,2,0) | $8 + C(\tau_2,(0,0,0)) + C(\tau_3,(0,1,1)) = 8 + 18 + 25 = 51$ |
| (0,2,1) | $6 + 2 + 8C(\tau_2,(0,0,0)) + C(\tau_3,(0,2,2)) = 16 + 18 + 23 = 57$ |
| (0,2,2) | $16 + C(\tau_2,(0,0,0)) + C(\tau_3,(0,2,2)) = 16 + 18 + 23 = 57$ |
| (1,0,0) | $6 + 2 + C(\tau_2,(2,0,0)) + C(\tau_3,(0,1,1)) = 8 + 14 + 25 = 47$ |
| (1,0,1) | $6 + 2 + 2 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,1,1)) = 10 + 11 + 25 = 46$ |
| (1,0,2) | $6 + 2 + 8 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,1,1)) = 16 + 11 + 25 = 52$ |
| (1,1,0) | $6 + 2 + 2 + C(\tau_2,(2,0,0)) + C(\tau_3,(0,1,1)) = 10 + 14 + 25 = 49$ |
| (1,1,1) | $6 + 2 + 2 + 2 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,2,2)) = 12 + 11 + 23 = 46$ |
| (1,1,2) | $6 + 2 + 2 + 8 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,2,2)) = 18 + 11 + 23 = 52$ |
| (1,2,0) | $6 + 2 + 8 + C(\tau_2,(2,0,0)) + C(\tau_3,(0,1,1)) = 16 + 14 + 25 = 55$ |
| (1,2,1) | $6 + 2 + 2 + 8 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,2,2)) = 18 + 11 + 23 = 52$ |
| (1,2,2) | $6 + 2 + 16 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,2,2)) = 24 + 11 + 23 = 58$ |
| (2,0,0) | $8 + C(2,(\tau_2,0,0)) + C(\tau_3,(0,1,1)) = 8 + 14 + 25 = 47$ |
| (2,0,1) | $6 + 2 + 8 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,1,1)) = 16 + 11 + 25 = 52$ |
| (2,0,2) | $16 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,1,1)) = 16 + 11 + 25 = 52$ |
| (2,1,0) | $6 + 2 + 8 + C(\tau_2,(2,0,0)) + C(\tau_3,(0,1,1)) = 16 + 14 + 25 = 55$ |
| (2,1,1) | $6 + 2 + 2 + 8 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,2,2)) = 18 + 11 + 23 = 52$ |
| (2,1,2) | $6 + 2 + 16 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,2,2)) = 24 + 11 + 23 = 58$ |
| (2,2,0) | $16 + C(\tau_2,(2,0,0)) + C(\tau_3,(0,1,1)) = 16 + 14 + 25 = 55$ |
| (2,2,1) | $16 + 6 + 2 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,2,2)) = 24 + 11 + 23 = 58$ |
| (2,2,2) | $24 + C(\tau_2,(2,0,2)) + C(\tau_3,(0,2,2)) = 24 + 11 + 23 = 58$ |

TABLE 1.4 – Les états possibles du nœud 1. Puisqu'on sait que $e_o = (1,1,1)$, on peut immédiatement conclure que la solution optimale est obtenu avec $C^*(1,(1,1,1)) = 43$. Elle nous est donnée par la solution de la figure 1.12.

$$
\begin{aligned}
& C^*(\tau_1,(1,1,1)) \\
= \;& C(\tau_1,(0,0,0)) \\
= \;& C^*(\tau_2,(0,0,0)) + C^*(\tau_3,(0,0,0)) \\
= \;& C(\tau_2,(0,0,0)) + C(\tau_3,(0,1,1)) \\
= \;& C^*(\tau_4,(0,0,0)) + C^*(\tau_5,(0,0,0)) + C^*(\tau_6,(0,1,1)) + C^*(\tau_7,(0,1,1)) + C^*(\tau_8,(0,1,1)) \\
= \;& C(\tau_4,(1,0,1)) + C(\tau_5,(1,0,0)) + C(\tau_6,(0,2,0)) + C(\tau_7,(0,2,2)) + C(\tau_8,(0,2,2))
\end{aligned}
$$

TABLE 1.5 – La relation de récurrence permettant de trouver la solution optimale.

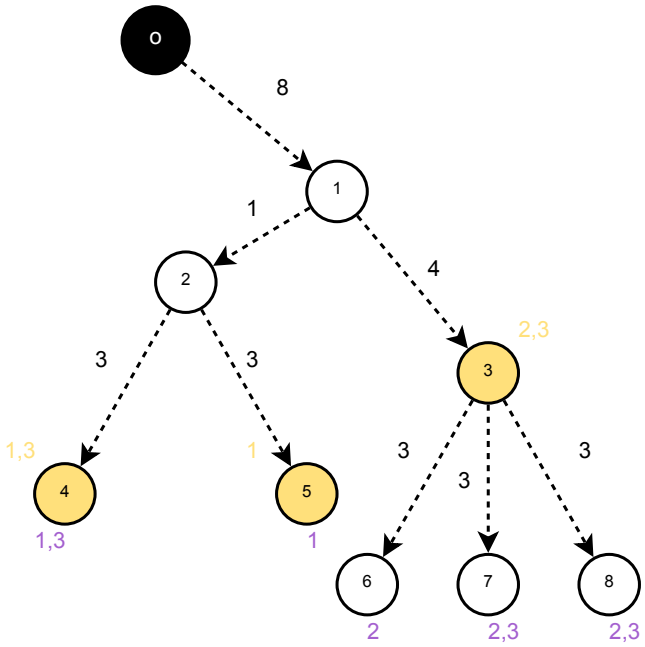



FIGURE 1.12 – *Solution optimale, retrouvée par "backtracking" des sous-états optimaux.*

L'auteur conclut alors sur la faiblesse de cette approche, qui en pratique ne permet pas de résoudre des instances de grande taille de par la complexité combinatoire de l'algorithme dynamique. En effet, il n'est pas garanti que cette procédure ne doive investiguer tous les états possibles des nœuds, même si les ensembles de faisabilité peuvent grandement contraindre cette exploration exhaustive. L'auteur ne donne pas de borne de complexité de son algorithme, mais suppose que celle-ci est exponentielle.

Les limites des hypothèses de [72], sont également liées au mode de transmission. Le multi-cast sous-entend une transmission des données qui est faite au même moment pour tous les utilisateurs, soit que l'ensemble des requêtes formulées soient effectuées quasiment au même moment. Ce n'est pas le cas en pratique, à plus forte raison pour de la vidéo à la demande où le but est de proposer un service au moment désiré par le client. Cela signifie qu'il est peu probable que deux utilisateurs requièrent la même vidéo au même moment, et qu'un décalage temporel, même minime, génère un nouveau trafic pour la totalité de chaque vidéo au sein du réseau. De plus, nous pourrions aisément conclure que le déploiement de caches transparents n'aurait que très peu de sens si l'hypothèse de simultanéité des requêtes s'avérait exacte, ce dispositif tirant justement parti de la succession temporelle des requêtes.

## Conclusion

Les problèmes de localisation sont des problèmes récurrents dans le domaine de la recherche opérationnelle. Même s'il en existe des faciles (reformulation par des plus courts chemins, arbre couvrant de poids minimum...), la plupart sont difficiles, et c'est pourquoi une littérature abondante existe à ce sujet, et beaucoup de méthodes ont été proposées pour résoudre certains cas

particuliers. Une des difficultés émergentes majeures est que, chaque variation du problème est susceptible d'apporter une nouvelle complexité et/ou des spécificités ; ceci a pour effet de rendre difficile la recherche d'une méthode générique et efficace pour tous ces problèmes.

Dans le cadre du déploiement de caches transparents, nous devons modéliser son comportement autonome particulier. De plus, ce déploiement s'effectue dans un réseau existant et au routage déterminé, ce qui nous place dans un réseau arborescent. Ces deux particularités nous ont conduits à nous intéresser à la programmation dynamique pour la localisation dans les arborescences. Nous avons proposé de décrire plus en détail deux approches existantes, qui traitent de problèmes de localisation sous des hypothèses différentes. Nous avons observé les forces et les limites de ces approches, ainsi que des mécanismes qui vont nous servir de base pour proposer une nouvelle approche adaptée à notre problème.

Mais avant de passer au problème particulier de localisation de caches transparents (chapitre 3), nous allons étudier un cas particulier de déploiement de CDN : le problème de localisation de dépôts et d'installation de liens à moindre coût, dans sa version $k$-médian. Ce problème est une variante des problèmes simples que nous avions énoncés. Le chapitre suivant va nous permettre de déterminer la complexité de ce problème ainsi qu'une approche pour le résoudre.

# Chapitre 2

# Localisation de $k$ caches/serveurs dans un réseau quelconque, approche par la théorie des matroïdes

## Introduction

Dans un contexte de télécommunication et plus précisément dans un déploiement de CDN, on peut être intéressé à vouloir construire un réseau de distribution de contenus avec la possibilité d'installation de serveurs et également de liens (ou amélioration de liens).

Dans le chapitre précédent nous avons déjà évoqué des types de problème de localisation simples. Celui dont nous allons parler ici est assimilable à un problème de localisation de dépôts et de liens. Les dépôts correspondent ici à des serveurs, tandis que les liens représentent l'installation, ou l'augmentation de capacité d'une connexion existante pour le trafic multimédia. Dans la pratique des télécommunications il est recommandé d'installer des liaisons systématiquement dans les deux sens, indépendamment du dispositif que nous voulons rattacher. Ceci pour deux raisons majeures : d'abord le coût supplémentaire pour créer la liaison inverse est très faible, la seconde est qu'il est souhaitable que la requête d'un contenu transite dans le lien dans l'autre sens. Ceci nous motive dès lors à considérer des graphes non orientés (ceci suppose une capacité non limitante installée dans les deux sens).

Dans la réalité l'opérateur peut avoir une contrainte sur le nombre de serveurs qu'il se permet de déployer. C'est le cas notamment quand la stratégie d'investissement préfère limiter les coûts de maintenance nécessaires pour ces serveurs dans un futur plus ou moins proche (les câbles réseau ayant une durée de vie généralement plus élevée que les terminaux informatiques puisque ces derniers sont moins soumis à l'obsolescence et aux pannes). Cette contrainte supplémentaire génère une variation du problème initial : c'est ce qu'on appelle la version $k$-médian ($k$ étant le nombre maximal de serveurs à déployer).

## Modèle de localisation, coûts fixes et arbre de poids minimal

Nous allons proposer ici une formulation du problème.

L'infrastructure actuelle munie des liaisons susceptibles d'être construites, est représentée par un graphe $G = (V, E)$. Chaque arête $e \in E$ est munie d'un coût d'ouverture (ou d'installation) $w_e \in \mathbf{R}^+, \forall e \in E$. Nous utiliserons de façon indifférenciée $e$ et $(ij)$ pour désigner une arête.

Les serveurs sont capables d'intercepter la totalité des requêtes. Nous appelons $o$ le nœud virtuel particulier additionné au graphe $G$ vers lequel émanent toutes les données du réseau. Il est relié par une arête vers chacun des serveurs déjà physiquement installés (éventuellement aucun), pour un coût nul.

Les variables $x_e, \forall e \in E$ représente l'installation effective d'une arête si sa valeur est 1, 0 sinon. A noter qu'avec la création du nœud virtuel $o$, cette arête peut aussi bien représenter une connexion de télécommunication, que l'installation d'un serveur. On appelle $G(V, X_E)$ le graphe privé des arêtes $e \in E$, si $x_e = 0$. On appelle également $p^{ij}$ la chaîne reliant $i$ à $j$, définie par son ensemble d'arêtes.

Le modèle peut alors s'écrire de la manière suivante :

$$\text{Localisation de serveurs 2} \begin{cases} \min \sum_{e \in E} x_e.w_e, & (2.1) \\ \exists p^{ij} \in G(V, X_E), \qquad \forall (i,j) \in V^2, & (2.2) \\ x_e \in \{0,1\} \qquad\qquad \forall e \in E & (2.3) \end{cases}$$

L'objectif est ici de minimiser le coût de déploiement du CDN (fonction objectif (2.1)), sous la contrainte que le graphe induit par les variables $x_e$ est connexe (2.2) : on doit atteindre chaque nœud du réseau.

Comme nous l'avions déjà annoncé dans le chapitre précédent, ce problème peut se reformuler comme un problème d'arbre couvrant de poids minimum. Ce problème bien connu est facile et peut se résoudre par un algorithme combinatoire fortement polynomial comme celui proposé par Kruskal ou Prim [62].

Or, nous avons une contrainte sur le nombre de serveurs que nous pouvons déployer, déterminé par $k$. Dans le cadre d'une intégration de cette limitation dans la reformulation de notre problème en arbre couvrant de poids minimum, cela se traduira tout simplement par une contrainte sur le degré de $o$ dans le graphe $G(V, X_E)$.

Nous allons étudier ce nouveau problème depuis sa complexité jusque dans la manière de le résoudre, en mobilisant notamment la théorie des matroïdes.

## 2.1 Modèle de localisation de caches/serveurs et $k$ médian

Nous pouvons maintenant reprendre notre modèle de localisation précédent, et nous allons à présent y inclure la contrainte de $k$-médian. Dans le cadre de la formulation par arbre couvrant de poids minimal, il s'agit de la contrainte sur le degré du nœud $o$ :

$$\text{Localisation de } k \text{ serveurs} \begin{cases} \min \sum_{e \in E} x_e.w_e, & (2.4) \\ \exists p^{ij} \in G(V, X_E), \qquad \forall (i,j) \in V^2, & (2.5) \\ \sum_{\substack{oi \in E \\ i \in V - \{o\}}} x_{oi} \leq k, & (2.6) \\ x_e \in \{0,1\} \qquad \forall e \in E & (2.7) \end{cases}$$

L'objectif est ici de minimiser le coût de déploiement du CDN (fonction objectif (2.4)), sous la contrainte que le graphe induit par les variables $x_e$ est connexe (2.5), et que l'on a installé un maximum de $k$ caches (2.6).

Nous aimerions alors naturellement savoir quelle est la complexité de ce problème, et quelles méthodes et/ou algorithmes sont envisageables pour le résoudre. Nous allons alors nous intéresser à la théorie des matroïdes, et notamment ses contributions dans le domaine de l'optimisation combinatoire.

## 2.2 Matroïdes graphiques

Introduit par Whitney ([104]) en 1935, le concept de Matroïde s'applique dans certains problèmes combinatoires ([31]). Il caractérise notamment les relations de dépendance entre les vecteurs colonnes d'une matrice, et permet d'exhiber des propriétés intéressantes comme l'optimalité des approches gloutonnes.

**Définition 6.** *Soit $E$ un ensemble fini et soit $\mathcal{F}$ une famille de sous-ensembles de $E$. Le couple $(E, \mathcal{F})$ est un matroïde si les propositions suivantes sont vérifiées :*

1. $\emptyset \in \mathcal{F}$
2. *Si $F \in \mathcal{F}$ et $F' \subseteq F$ alors $F' \in \mathcal{F}$*
3. *Si $F, F' \in \mathcal{F}$ avec $|F'| = |F| + 1$, alors il existe $e \in F' - F$ tel que $F \cup \{e\} \in \mathcal{F}$*

*Les éléments de $\mathcal{F}$ sont alors appelés ensembles indépendants de $E$.*

Nous rappelons également quelques notions les concernant :

**Définition 7.** *Soit $(E, \mathcal{F})$ un matroïde et soit $E' \subseteq E$. Le rang de $E'$, noté $rg(E')$ est la cardinalité du plus grand sous-ensemble indépendant dans $E'$ ;*

$$rg(E') = \max_{F \subseteq E', F \in \mathcal{F}} |F| \tag{2.8}$$

*Le rang de $E$ est le rang du matroïde $(E, \mathcal{F})$.*

**Définition 8.** *Soit $(E, \mathcal{F})$ un matroïde et soit $E' \subseteq E$. La fermeture de $E'$, que l'on note $cl(E')$ est le plus grand sous-ensemble $E'' \supseteq E'$ de $E$ tel que $rg(E'') = rg(E')$*

Sachant que ;

**Théorème 7.** *Soit $(E, \mathcal{F})$ un matroïde et $E' \subseteq E$. La fermeture $cl(E')$ est unique.*

**Définition 9.** *Soit $(E, \mathcal{F})$ un matroïde. Un sous-ensemble $E' \subseteq E$ est un stigme si $E' \notin \mathcal{F}$ et $E' - \{e\} \in \mathcal{F}$ pour tout $e \in E'$*

En rappelant que ;

**Théorème 8.** *Soit $(E, \mathcal{F})$ un matroïde. Si $F \in \mathcal{F}$ et $F \cup \{e\} \notin \mathcal{F}$ pour $E - F$, alors $F \cup \{e\}$ contient un unique stigme.*

Un théorème important en optimisation découle des matroïdes :

**Théorème 9.** *Soit $(E, \mathcal{F})$ un matroïde et soit $c : E \to \mathbb{R}^+$ une fonction de coût quelconque. L'algorithme suivant :*

**Algorithm 4** Algorithme Glouton

```
1: Ordonner les éléments de E de telle sorte que E = {e_1, e_2, ..., e_n} avec c(e_1) ≥ ... ≥ c(e_n)
2: Poser F ← ∅
3: for all i ∈ [[1, n]] do
4:    if F ∪ {e_i} ∈ F then
5:       F ← F ∪ {e_i}
6:    end if
7: end for
```

*produit un ensemble $F \in \mathcal{F}$ de coût $c(F)$ maximum*

Soit un graphe non orienté $G = (V, E)$, et $\mathcal{F}$ l'ensemble des sous-graphes partiels sans cycle. On peut voir alors que les deux premières propriétés des matroïdes sont vérifiées. De plus, on sait que pour un sous-graphe $G' = (V, E')$ avec $E' \subseteq E$, son ensemble maximal sera nécessairement toujours égal à $|V| - g(G')$, avec $g(G')$ le nombre de composantes connexes dans $G'$. On peut donc conclure que $(E, \mathcal{F})$ est un matroïde, qui est généralement appelé matroïde graphique.

Lorsque l'on veut générer un arbre couvrant sur $G = (V, E)$, on cherche de manière équivalente à générer le plus grand sous-graphe partiel qui ne contient pas de cycle. Il s'agit donc de trouver un ensemble maximal $F$ sur $E$, avec $F \in \mathcal{F}$.

L'arbre couvrant de poids minimum consiste à trouver un ensemble maximal qui minimise $c(F)$. En ordonnant les coûts de manière croissante, nous pouvons dès lors utiliser un algorithme glouton pour résoudre ce problème ; nous retrouvons ainsi l'algorithme de Kruskal ([62]).

## 2.3 Matroïde de degré $k$ sur un nœud

Cependant, dans notre problème une contrainte supplémentaire est émise sur le degré de $o$. Nous pouvons exhiber un exemple qui montre dès lors que le matroïde graphique explicité, additionné de cette contrainte, n'est plus un matroïde.

Soit la famille $\mathcal{F}'$ des sous-graphes partiels sans cycle avec $deg(o) \leq k$. La figure 2.1 nous montre un exemple qui prouve que $(E, \mathcal{F}')$ n'est pas un matroïde. $F_1$ et $F_2$ (représenté par les arêtes pleines) sont tout deux des sous-graphes partiels sans cycles, dont le degré sur le nœud $o$ est inférieur ou égal à 2. On remarque que $|F_1| + 1 = |F_2|$. L'ensemble $F_2 - F_1$ est $\{(ao), (bd)\}$. On remarque que si l'on ajoute $(ao)$ à $F_1$, la contrainte de degré est violée, tandis que si l'on ajoute $(bd)$ à $F_1$, c'est la contrainte d'acyclicité qui est violée. Il n'existe donc pas d'élément (d'arête) $e \in F_2 - F_1$ telle que $F_1 \cup \{e\} \in \mathcal{F}$, ce qui contredit la troisième propriété constitutive d'un matroïde.

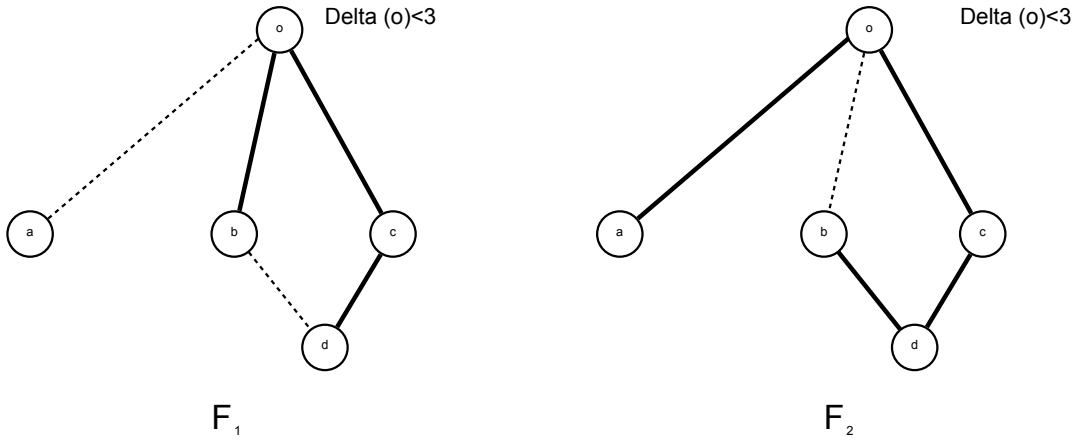



FIGURE 2.1 – Exemple qui prouve que notre problème n'est pas un matroïde.

Nous allons à présent définir le couple $(E, \mathcal{F}_k)$, avec $\mathcal{F}_k$ l'ensemble des sous-graphes tels que $deg(o) \leq k$.

**Proposition 10.** *$(E, \mathcal{F}_k)$ est un matroïde.*

*Démonstration.* Nous remarquons les propriétés suivantes :

1. (1) $\emptyset \in \mathcal{F}_k$
2. (2) Si $F \in \mathcal{F}_k$ et $F' \subseteq F$ alors $F' \in \mathcal{F}_k$

Considérons à présent deux ensembles $F, F' \in \mathcal{F}_k$, avec $|F| = |F'| + 1$. Soit $\Delta = F - F'$, et $deg(o, F)$ le degré du nœud $o$ dans le sous-graphe partiel $F$ (respectivement $F'$). Sachant que $deg(o, F) \leq k$ et $deg(o, F') \leq k$, nous pouvons distinguer les cas suivants :

1. $deg(o, F) > deg(o, F')$ : alors il existe un élément $e = (oi)$ de $\Delta$ qui peut être ajouté à $F'$ sans violer la contrainte de degré pour $F'$ qui n'est pas serrée.
2. $deg(o, F) \leq deg(o, F')$ : alors il existe un élément $e \in \Delta, e \neq (oi) \forall i \in V$ qui peut être ajouté à $F'$, sans influencer le degré de $o$ pour $F'$

Nous validons ainsi la propriété (3) d'un matroïde. □

## 2.4 Algorithme d'intersection de deux matroïdes

Soient deux matroïdes $(E, \mathcal{F}_1)$ et $(E, \mathcal{F}_2)$ et une fonction de poids $c : E \to \mathbb{R}$. Pour un entier donné $n$, le problème d'intersection de matroïdes consiste à trouver (s'il existe) l'ensemble $F_n*$ indépendant commun ($F_n* \in \mathcal{F}_1 \cap \mathcal{F}_2$) de $n$ éléments de poids maximal.

J. Edmonds démontra qu'il était possible de résoudre ce problème de manière optimale et polynomiale, et proposa notamment un algorithme générique pour le résoudre [31]. D'autres auteurs modifièrent/améliorèrent par la suite ces approches comme ([41] et [63]).

**Principe général**

L'idée générale de l'algorithme est d'ajouter successivement un élément de coût maximal qui respecterait l'appartenance aux deux familles, tant qu'aucun rang de matroïde n'est atteint par cet ensemble. Comme il est probable que ce ne soit pas toujours possible, l'algorithme détermine le cas échéant quelle permutation neutre (au sens des coûts de chaque matroïde) permettrait d'ajouter cet (ou ces) élément(s) de coût maximaux. Enfin, dans le cas où même cette transformation est impossible, comment modifier ces coûts pour déterminer l'(es) élément(s) suivant(s) qui pourrai(en)t être intéressant(s).

En notant, pour tout ensemble $F \subseteq E, F \in \mathcal{F}$, $cl_i(F)$ la fermeture de $F$ dans le matroïde $i$ (unique d'après 2.2), et $S_i(x, F)$ le stigme induit par $x$ sur l'ensemble $F$ dans le matroïde $i$ (unique d'après 2.2). Le pseudo-code de l'algorithme est présenté par le pseudo-code 5. Nous en donnons une description ci-après :

Soit (1) désignant le premier matroïde et (2) le second. L'algorithme se présente comme suit ; nous initialisons notre ensemble d'arêtes comme vide, soit à l'itération 0, $F_0 = \emptyset$ (étape 1). Les éléments sont munis d'un coût différent pour chaque famille. Pour l'une des familles il sera initialisé par le coût réel de l'arc (coût dans le problème), tandis que pour l'autre, il sera initialisé à 0. Nous choisissons le matroïde (2) pour supporter les coûts "réels" et (1), les coûts nuls. Nous allons ensuite répéter les étapes suivantes tant que la cardinalité de notre ensemble $F_n$ n'excède pas le rang de l'un de nos deux matroïdes (répétition des étapes de 3 à 25 décrites ci-après).

Nous calculons **pour chaque famille**, l'ensemble $E_i$ ($i \in \{(1), (2)\}$ désignant l'un des deux matroïdes) des éléments de coût maximal (car un élément de coût maximal n'est pas nécessairement unique), qui ne sont pas encore compris dans notre ensemble $F$, et tels qu'ils formeraient toujours une famille de ce matroïde si ils sont ajoutés (étapes 4 et 5). Le lecteur remarquera que cela correspond donc à l'ensemble complémentaire de la fermeture de $F_n$ dans $E$ pour ce matroïde, et dont les coûts sont maximaux.

Nous allons extraire le graphe **orienté** suivant : $G(E, A)$ où les nœuds du graphes représentent les éléments de $E$, et où les arcs $A$ représentent une interversion possible de ces éléments sur notre ensemble courant $F_n$. Le lecteur pourra porter une attention particulière des appellations ; lorsque nous parlerons d'arêtes il s'agira ou des liens du graphe "réel" $G(V, E)$, ou des nœuds du graphe de $G(E, A)$. Les arcs, en revanche ne désigneront que les liens dans le graphe de $G(E, A)$. L'ensemble des arcs est initialisé comme vide (étape 7). Nous créons alors les arcs de la manière suivante.

Pour le matroïde (1), nous regardons chaque élément $x$ qui ne peuvent être ajoutés à $F_n$ sans rompre la définition de la famille pour ce matroïde. Le lecteur notera donc que cela correspond aux éléments de la fermeture engendrée par $F_n$ et qui ne sont pas dans $F_n$. Nous regardons alors pour chacun de ces $x$, quels sont les éléments $y_n$ de $F_n$ qui peuvent être intervertis avec $x$ de sorte que $F_n$ reste une famille pour le matroïde (1), et qui possèdent un coût identique. Le lecteur remarquera alors qu'il s'agit du stigme engendré par $F_n \cup \{x\}$ privé de $x$ et dont les coûts sont identiques. Nous construisons alors les arcs pour chacun de ces couples $(x, y)$.

Pour le matroïde (2), nous générons des arcs similaires mais dans un sens opposé. C'est-à-dire que nous regardons les éléments qui pourraient être ajoutés en retirant un autre élément de même coût selon le matroïde 2, de sorte à ce que l'on obtienne toujours un ensemble appartenant à la famille de ce matroïde.

On pourra alors voir les arcs comme des possibilités de modification de notre ensemble $F_n$ par échanges d'éléments avec son complémentaire, qui conservent la propriété de famille, et qui garde le même coût. De ce fait, si les meilleurs éléments (en terme de coût) selon chaque famille de matroïdes peuvent être reliés par au moins un chemin, il existe une manière de les ajouter tous les deux (et retirer celui qui était dans $F_n$), sans modifier le coût de la solution courante, et en gardant la propriété d'appartenance aux deux familles. Ceci est représenté par l'étape de 11 à 15, et également par la figure 2.2. Le chemin possède deux éléments non inclus dans le $F_n$ courant : le départ et l'arrivée. Les éléments intermédiaires sont ceux qui seront permutés. Le nouvel ensemble $F_{n+1}$ est donc naturellement la différence symétrique entre $F_n$ et ce chemin.

Dans le cas où il n'est pas possible de relier ces deux éléments, nous allons alors changer les coûts. Comme nous cherchons à maximiser le coût de la solution, et que nous avons défini le point de "départ", les éléments non atteignables doivent avoir un coût plus important pour devenir "intéressants". Ceci dans le but de créer des liens et/ou que les ensembles $E_1$ et $E_2$ aient une intersection non nulle. Cependant, pour s'assurer qu'il y ait **conservation du coût total**, il est nécessaire de faire basculer ce coût d'un matroïde à l'autre.

La valeur de coût ainsi "basculée" doit être prise selon le minimum des différentiels entre l'ensemble des arêtes "atteignables" et l'ensemble des arêtes "non-atteignables" depuis $E_2$ (ceci dans le but de contrôler la "pénalité", et s'assurer que nous ne construisons pas un problème opposé). C'est ce qui est calculé dans les étapes 16 à 20, avant de modifier réellement les coûts des éléments pour chacun des matroïdes (étapes 21 à 24).

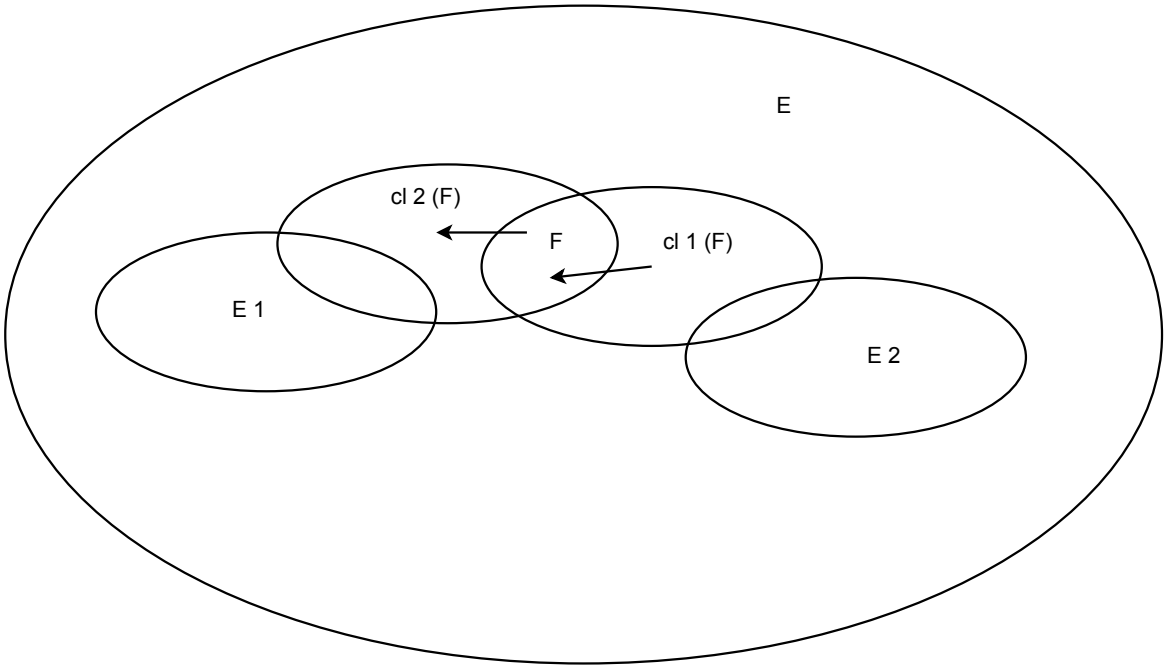



FIGURE 2.2 – Représentation des ensembles induits par les deux Matroïdes et le graphe $G = (E, A)$

Dans le cas général, cet algorithme possède une complexité donnée par $O(|E|^3.\theta)$, où $\theta$ représente la complexité maximale entre les deux oracles d'indépendance (vérification de l'appartenance à la famille).

Il est intéressant de noter que le problème d'intersection de $\mathcal{N}$ matroïdes, $\mathcal{N} > 2$, est NP-difficile.

## 2.5 Algorithme de localisation de $k$ caches et déploiement de CDN

Nous appelons ici $M_g = (E, \mathcal{F}_g)$ et $M_k = (E, \mathcal{F}_k)$, qui désignent respectivement le matroïde de sous-graphes partiels acycliques, et le matroïde de sous-graphes tels que $deg(o) \leq k$. La fonction de coût est la fonction $c : E \to \mathbb{R}^+$ correspondant à l'installation d'une arête (l'installation des caches correspond également, sur le graphe, à l'installation d'une arête). Nous appelons $\mathcal{E}_o$ l'ensemble des arêtes $oi \in E, i \in V - \{s\}$.

Pour construire notre algorithme de résolution exact et polynomial, nous allons appliquer l'algorithme général d'intersection de deux matroïdes, en effectuant le maximum de simplifications. Pour ce faire nous allons commencer par exhiber quelques propriétés.

Le lecteur remarquera qu'il s'agit ici d'un problème de minimisation du coût de l'ensemble indépendant maximal, et non de maximisation du coût de l'ensemble maximal. Il conviendra naturellement, d'adapter l'algorithme en conséquence.

### 2.5.1 Propriétés particulières

Nous introduisons quelques propriétés qui nous serviront à réduire le nombre d'étapes nécessaires à notre algorithme. Les notations utilisées ici sont les mêmes que celles données pour l'algorithme général d'intersection de deux matroïdes.

**Proposition 11.** *La fermeture d'un ensemble $F_n$ sur le matroïde $M_k$ sera égale à $F_n$ ou $CL_k = \mathcal{E}_o$. Une fois la fermeture égale à $CL_k$ à une itération $n$, la fermeture reste constante à une itération $m > n$.*

*Démonstration.* Nous remarquons tout d'abord que pour le matroïde $M_k$, la fermeture d'un $Cl_k(F)$ pour un ensemble $F \in \mathcal{F}$ donné est égale à $F$, tant que la contrainte de degré n'est pas serrée ($deg(o) < k$). Cela signifie qu'aucun arc de $G = (E, A)$ pour le matroïde $M_k$ ne sera créé tant que $deg(o) < k$. Lorsqu'elle sera ensuite serrée ($deg(o) = k$), la fermeture sera connue : il s'agira tout simplement des arêtes de l'ensemble $\mathcal{E}_o$ (puisque l'on ne peut ajouter un autre élément $e \in \mathcal{E}_o$ sans violer la contrainte de degré, tandis que les autres arcs ne sont pas concernés par cette contrainte).

Les arcs lient deux éléments à l'intérieur de la fermeture. L'utilisation de l'un d'eux générera une permutation à l'intérieur de la fermeture, cette dernière ne changera donc pas.

La fermeture restera donc un ensemble constant jusqu'à la fin de l'algorithme, quel que soit $F_n$, nous appellerons alors cet ensemble $CL_k$. On aura dès lors $\mathcal{E}_o = CL_k$. □

**Proposition 12.** *Au cours de l'algorithme, tant que le contrainte $deg(o) < k$ et $E_g \neq \emptyset$, il existe un élément $e \in E_g \cap E_k$.*

*Démonstration.* Avant que la contrainte de degré soit serrée tous les coûts $c_k(e)$ seront nuls et identiques, ce qui signifie, d'après la propriété (2.5.1) que $E_k = E - F_k$. On a donc $E_g \subseteq E_k$, ce qui veut dire que l'intersection de ces ensembles est non nulle tant que $E_g \neq \emptyset$. □

**Proposition 13.** *Les coûts $c_k(e), e \in E - \mathcal{E}_o$, resteront nuls à chaque itération.*

*Démonstration.* Les coûts $c_k(e), e \in E$ sont initialement nuls, et le restent jusqu'à ce que la contrainte $deg(o) \leq k$ soit serrée (d'après 12). Si l'on nomme $H$ l'ensemble des arêtes atteintes depuis $E_g$, on remarque qu'il ne peut pas contenir une partie de $E - CL_k$ sans qu'un chemin existe de $E_g$ à $E_k$, car $c_k(e) = 0, \forall e \in E - CL_k$. Ceci signifie aucun changement de coût $c_k(e)$ pour les éléments $E - CL_k$. Par récurrence, cette assertion restera vraie jusqu'à la fin de l'algorithme. □

**Remarque 1.** *De manière analogue, les poids $c_g(e), \forall e \in E - CL_k$ ne changeront pas (leurs changements peuvent uniquement survenir si le coût de l'élément $e$ selon le second matroïde diffère également).*

**Proposition 14.** *Une fois la contrainte $deg(o) \leq k$ serrée, à toute itération $E_k = E - \mathcal{E}_o$.*

*Démonstration.* Une fois la contrainte serrée, il est impossible de choisir un élément $e$ de $\mathcal{E}_o$ sans violer la contrainte, et donc respecter l'appartenance à la famille du matroïde $M_k$. D'après la propriété 13, les coûts $c_k(e), e \in E - \mathcal{E}_o$ seront tous égaux et nuls à toute itération. Cela signifie que l'on peut choisir un des éléments de cet ensemble de manière indifférenciée. □

**Remarque 2.** *Ceci est particulièrement intéressant, car si nous ne voulons prendre qu'un élément de $E_k = E - \mathcal{E}_o$. Si l'un est atteignable, on se fiche d'en atteindre un autre : il n'est dès lors pas nécessaire de construire les arcs selon le matroïde $M_g$ qui relient deux arêtes à l'intérieur de cet ensemble.*

### 2.5.2 Simplifications

Nous allons donc pouvoir supprimer/raccourcir des étapes de l'algorithme général. Dans un premier temps, nous pourrons appliquer l'algorithme de Kruskal jusqu'à ce que la contrainte de degré soit serrée d'après la propriété 12. Par la suite, si un élément "préféré" au sens des coûts par le matroïde graphique $M_g$ ne fait pas partie de l'ensemble $\mathcal{E}_o$, nous savons que cet élément peut être rajouté dans notre ensemble $F_n$ d'après la propriété 13.

Une fois la contrainte serrée, les arcs selon le matroïde de degré $M_k$, seront construit systématiquement à partir des arêtes $\mathcal{E}_o$ qui n'ont pas été pris vers ceux qui ont été pris dans $F_n$, et qui possèdent un coût $c_k$ identique. L'algorithme pour créer les arcs d'un élément $e$ est résumé dans le pseudo-code 6. Sa complexité est donnée par $O(k)$ (nombre d'arêtes de $\mathcal{E}_o$ prises dans $F_n$ à tester).

D'autre part, les arcs de $M_g$ seront construits à partir des arêtes $F_n$ incluses dans $\mathcal{E}_o$ (d'après 2), vers celles qui doivent être retirées pour ne pas former de cycle. L'algorithme pour créer les arcs d'un élément $e$ est résumé dans le pseudo-code 7. Sa complexité est donnée par $O(|E|)$ (test des $|E|$ arêtes qui peuvent recréer une composante connexe acyclique —stigme).

On remarque alors qu'une fois la contrainte serrée l'ensemble des arêtes $\mathcal{E}_o \cap F_n$ vont former une sorte de "hub" par lequel d'éventuels chemin vont passer. Il nous suffit donc de créer pour chaque élément $\mathcal{E}_o \cap F_n$ l'ensemble $E^+(e)$ des arêtes "prédécesseurs", et $E^-(e)$ l'ensemble des

"successeurs".

Reprenons à présent l'algorithme principal, et ce qu'il devient dans 8. Nous devons ajouter un élément tant que cela est possible (répétition des étapes de 3 à 19). On recherche les éléments de coûts minimaux selon le matroïde $M_g$ qui peuvent être ajoutés sans générer de cycle (étape 3). Nous déterminons l'ensemble des éléments de coûts minimaux selon le matroïde $M_k$ qui peuvent être ajoutés sans violer la contrainte de degré (étape 4). Il s'agit soit de $E$ (contrainte non serrée) soit de $E - \mathcal{E}_o$ (contrainte serrée).

S'il existe un élément de l'intersection de ces deux ensemble (étape 5), nous l'ajoutons (étape 6 et 7).

Si par contre cette intersection est nulle, alors nous construisons le graphe $G = (E, A)$ à l'aide des algorithmes 7 et 6, pour chaque élément de $F_n$ appartenant à $\mathcal{E}_o$ (étape 9). Nous cherchons ensuite l'ensemble des arêtes qui peuvent être atteinte depuis l'ensemble des éléments de coûts minimum selon le matroïde $M_g$ (étape 10). Si nous pouvons créer un chemin vers un élément de $E - CL_k$, alors on effectue la transformation pour ajouter l'élément (différence symétrique de $F_n$ avec le chemin $p$) (étape 12). Sinon on "bascule" une partie des coûts de l'ensemble atteint, entre les deux matroïdes de façon à "pénaliser" le choix du matroïde graphique $M_g$ sur ces arêtes (étapes 14 et 15). Ce différentiel $\delta$ est calculé à partir des algorithmes 7 et 6 comme le différentiel de coût (coût de l'arête d'arrivée moins le coût de l'arête de départ) des **arcs divergents de** $H$ tels **qu'ils auraient pu être créés par l'algorithme de construction d'arcs sans la contrainte d'égalité de coût** (étape 13). Le lecteur pourra remarquer le lien avec la notion de valeur de variable duale associée aux contrainte d'acyclicité et de degré $k$.

**Théorème 15.** *Notre algorithme possède une complexité de* $O(|E|.log(|E|) + (|E| - k).k.|E|)$.

*Démonstration.* La complexité de cet algorithme peut être calculé comme suit. Nous savons que nous devrons calculer le(s) meilleurs éléments à ajouter $e$ pour chaque matroïdes. La complexité minimale entre les deux matroïdes est donc $O(|E|log(|E|))$.

Ensuite nous devrons ensuite éventuellement créer le graphe $G = (E, A)$ au plus tôt après $k$ itérations. Cette construction nécessite $k$ appels pour chaque arête $e \in \mathcal{E}_o \cap F_n$. On a donc une complexité totale pour la création du graphe de $O(k.|E|)$, donnée par la complexité maximale de l'algorithme 7. Le plus court chemin par une méthode de breadth-first search peut être effectuée en $O(|E|)$, mais cette complexité est déjà dominée par la construction des arcs. Enfin, comme il restera au plus $|E| - k$ éléments à ajouter, soit autant d'itérations à faire, nous obtenons notre complexité de $O(|E|.log(|E|) + (|E| - k).k.|E|)$. □

**Remarque 3.** *Nous remarquons que pour les cas particuliers* $k = 0$ *ou* $k = |E|$*, nous retrouvons la complexité de l'algorithme de Kruskal. Le premier cas réduit tout simplement le problème avec un nœud en moins, et le second est une relaxation implicite de la contrainte de degré.*

## 2.6 Exemple

Nous présentons ici le déroulement de l'algorithme sur un exemple simple. L'instance est représentée dans la figure 2.3. La première phase consiste à appliquer l'algorithme de Kruskal tant qu'il existe un élément d'intersection de $E_k$ et $E_g$.

Ceci nous amène à la situation décrite par la figure 2.4. L'ensemble $F_n$ est représenté par les arêtes pleines. $E_g$ est ici le singleton de l'arête la moins coûteuse qui respecte l'acyclicité, représentée en bleu. $E_k$, représenté en rouge, est l'ensemble des arêtes non comprises ni dans $CL_k$, ni dans $F_n$, et qui sont toutes de coût minimal : $(ab)$ et $(bc)$. Puisque l'ensemble est disjoint, nous devons créer le graphe orienté $G = (E, A)$ des "permutations". Nous construisons alors les arcs du graphe $G(E, A)$ pour le matroïde $M_k$ qui sont représentés en rouge. Les arcs du graphe $G = (E, A)$ pour le matroïde $M_g$ sont, eux, représentés en bleu.

Nous remarquons qu'il existe alors un chemin possible (représenté en vert sur la figure 2.5) qui relie un élément de $E_g$ à $E_k$.

Nous effectuons la transformation de notre ensemble afin de construire $F_{n+1}$, il s'agit de la différence symétrique avec le chemin choisi. La figure 2.6 représente l'état de $F_n$ après modifications par le chemin. On remarque que l'arc $(ob)$, bien qu'intéressant en terme de coût, est partie intégrante de la fermeture des deux Matroïdes selon $F_n$.

A nouveau, les ensembles $E_g$ et $E_k$ sont disjoints, nous devons donc construire le graphe $G = (E, A)$, comme nous le montre la figure 2.7.

Nous remarquons cette fois que l'ensemble des arêtes atteignables depuis $E_g$ dans le graphe $G = (E, A)$, n'atteint aucun élément de $E_k$. La figure 2.8 nous montre les éléments atteignables $H$ (en vert), et les arcs de $G = (E, A)$ qui auraient pu être créés sans la contrainte de coût, et qui divergent de $H$. Le minimum de différentiel $\delta$ correspond ici au différentiel de l'arc "potentiel" reliant $(oc)$ à $(ab)$, qui est de 1.

Nous mettons à jour les coûts (figure 2.9). Il s'agit d'augmenter les coûts du matroïde $M_g$ par $\delta$ sur les éléments atteignables, et réduire d'autant les coûts du matroïde $M_k$ sur ce même ensemble. Nous remarquons que les ensembles $E_k$ et $E_g$ sont toujours disjoints.

Nous construisons le graphe $G = (E, A)$, il n'existe toujours pas de chemin de $E_g$ vers $E_k$ (figure 2.10). Nous devons donc modifier les coûts des arêtes vertes 2.11, ce $\delta$ est calculé à partir de l'arc en bleu. Ce qui nous permettra de construire le chemin atteignant cet arc immédiatement à l'itération suivante.

Nous avons obtenu le rang du matroïde $M_g$ (et par hasard, celui de $M_k$ également), l'algorithme est terminé (figure 2.12). La solution optimale possède un coût de 14.

## 2.7 Cas particuliers

Dans le cas où le coût d'installation pour chaque cache/serveur est identique, on pourra remarquer simplement que la pénalité/bonus $\delta$ qui sera appliqué(e) concernera nécessairement tout l'ensemble $CL_k$, lorsque la contrainte $deg(o) \leq k$ sera serrée (puisqu'ils seront tous compris dans $E_g$), et ce pour toute étape de l'algorithme.

Nous pourrions également regarder le problème de localisation de $k$ caches, où le nombre de caches à installer doit être **exactement** de $k$ ($deg(o) = k$). Une méthode pour résoudre le problème est alors d'imputer $-M$ à toutes les arêtes $(oi), i \in V - \{o\}$, avec $M = \max_{e \in \mathcal{E}_o} c(e)$. Il

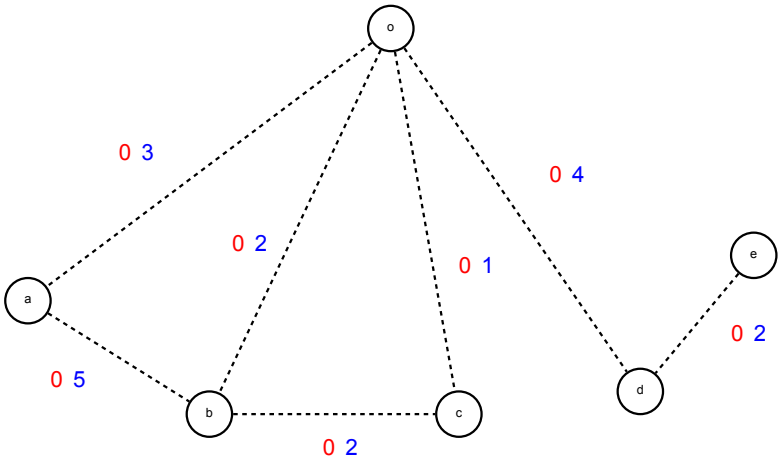



FIGURE 2.3 – Graphe sur lequel on doit trouver un arbre couvrant de poids minimum, avec une contrainte sur le degré $o$.

conviendra alors d'ajouter $k.M$ au coût de la solution optimale qui est extraite.

# Conclusion

Nous avons étudié ici le problème de localisation de $k$ serveurs et de liens dans un réseau de distribution de contenus. Nous avons montré qu'avec des hypothèses de coûts intensifs, nous pouvions formuler ce problème comme un arbre couvrant de poids minimal, avec contrainte sur le degré d'un sommet. Nous avons alors caractérisé la difficulté de notre problème en montrant qu'il s'assimilait à l'intersection de deux matroïdes, et que par conséquent, il pouvait être résolu par un algorithme fortement polynomial. Nous avons ensuite étudié en détail quelles étaient les propriétés particulières de notre problème qui pourraient réduire la complexité théorique du cas général. Ceci nous à permis d'extraire un algorithme fortement polynomial en $O(|E|.log(|E|)+(|E|-k).k.|E|)$ pour résoudre le problème de localisation de $k$ serveurs à moindre coût.

Comme nous l'avions déjà évoqué au chapitre 1, ce type de modèles n'est pas satisfaisant dans le cadre de la localisation de caches transparents, pour les raisons évoquées dans le chapitre précédent, il reste cependant pertinent dans la cas d'un déploiement d'un réseau total de CDN où aucune infrastructure n'a été installée. Ceci n'est pas notre cas.

Dans le chapitre 1, nous avions présenté particulièrement deux approches de programmation dynamique pour résoudre des problèmes de localisation de dispositifs dans les arborescences. Dans la première, le hit ratio d'un cache est considéré comme constant, ce qui n'est pas une hypothèse satisfaisante dans la pratique.

Dans la seconde approche, une précision est donnée aux types de contenus des dispositifs, et les flots peuvent être répliqués (ce qui en fait une grandeur intensive, ou "multi-cast"). Cette hypothèse est discutable surtout dans le contexte étudié (vidéo à la demande). De plus, dans ce problème le contenu du serveur est une variable de décision tandis que pour un cache, ce

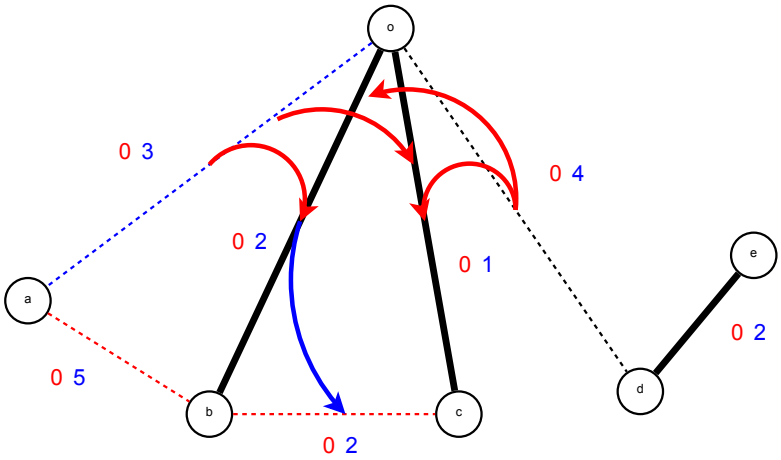



FIGURE 2.4 – Création du graphe des "permutations" $G = (E, A)$.

comportement est contraint.

Ces approches restent extrêmement intéressantes, car elles considèrent la programmation dynamique sous des points de vue sensiblement différents. Notre étude va consister à produire une nouvelle approche en s'inspirant de ces deux paradigmes, dans le but de traiter efficacement des problèmes de localisation génériques et difficiles.

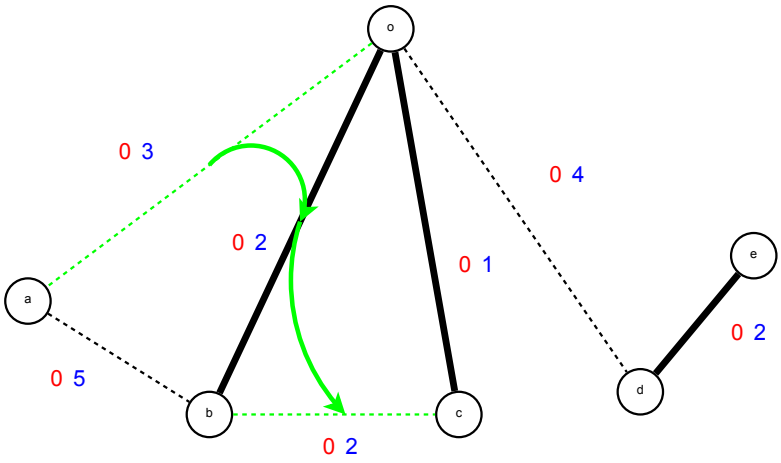



FIGURE 2.5 – Chemin possible reliant un élément de $E_g$ à $E_k$.

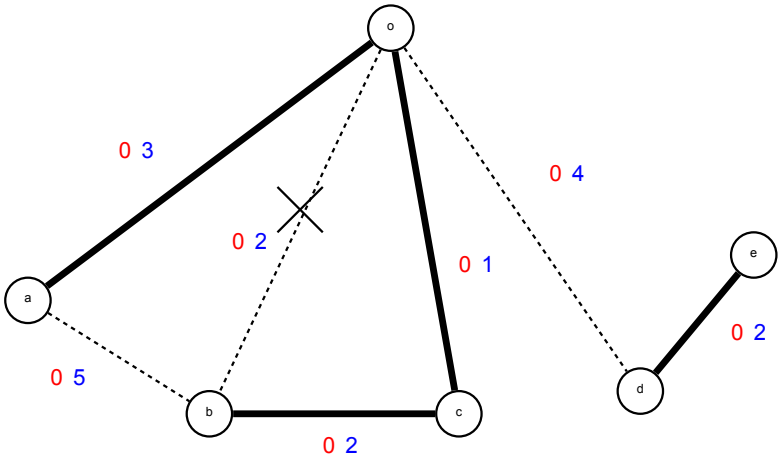



FIGURE 2.6 – Nouvel ensemble $F_n$.

**Algorithm 5** Algorithme d'intersection de deux matroïdes

**Require:** $M_1 = (E, \mathcal{F}_1)$, $M_2 = (E, \mathcal{F}_2)$, $G = (V, E)$ et fonction de coût $c : E \to \mathbb{R}$
**Ensure:** $F_n*$ ensemble indépendant de cardinalité $n$, de poids maximal

$n \leftarrow 0$, $F_0 \leftarrow \emptyset$, $c_2(e) \leftarrow c(e)$, $c_1(e) \leftarrow 0, \forall e \in E$
**while** $n < rg(M_1)$ et $n < rg(M_2)$ **do**
  **for all** $i$ : 1 et 2 **do**
    $m_i \leftarrow \max\{c_i(e), e \in E - cl_i(F_n)\}$ détermination de la valeur maximale $c(e)$ de l'élément non encore introduit, tel que $F_n \cup \{e\} \in \mathcal{F}_i$ (légal selon la famille $i$)
    $E_i \leftarrow \{e \in E - cl_i(F_n), c(e) = m_i\}$ ensemble des éléments $e$ tels que $F_n \cup \{e\} \in \mathcal{F}_i$, de valeur $m_i$ (valeur maximale)
  **end for**
  Création de $G = (E, A)$ graphe orienté, $A \leftarrow \emptyset$
  $A \leftarrow A \cup \{(xy), x \in cl_1(F_n) - F_n, y \in S_1(x, F_n) \cap F_n, c_1(x) = c_1(y)\}$ Dans le matroïde 1 ; arcs créés depuis chaque élément du complémentaire de $F_n$ dans la fermeture, vers l'ensemble des éléments du stigme qu'il induit dans $F_n$, si le coût de l'élément de départ est identique au coût de l'élément d'arrivée
  $A \leftarrow A \cup \{(xy), y \in cl_2(F_n) - F_n, x \in S_2(y, F_n) \cap F_n, c_2(x) = c_2(y)\}$ Dans le matroïde 2 ; arcs créés vers chaque élément de l'ensemble du complémentaire de $F_n$ dans la fermeture, depuis l'ensemble des éléments du stigme qu'il induit dans $F_n$, si le coût de l'élément de départ est identique au coût de l'élément d'arrivée.
  $H$ : ensemble des sommets de $G = (E, A)$ atteignables depuis $E_2$
  **if** $H \cap E_1 \neq \emptyset$ **then**
    Déterminer le plus court chemin $P \in H$ au sens des sommets de $G = (E, A)$ qui relie un élément de $E_2$ à $E_1$
    $F_{n+1} \leftarrow F_n \Delta P$ Création de $F_{n+1}$, par l'ajout de l'élément $e_2$ de $E_2$, et la modification neutre (en terme de coûts) qu'il induit sur $F_k$ pour les deux matroïdes.
    $n \leftarrow n + 1$
  **else**
    $\delta_{cl1} \leftarrow \min\{c_1(y) - c_1(x), x \in (cl_1(F_n) \cap H) - F_n, y \in (S_1(x, F_n) \cap F_n) - H\}$ Le minimum de différentiel de coût créé par les arcs "potentiels" internes à la fermeture 1 ; et qui divergent de $H$
    $\delta_1 \leftarrow \min\{m_1 - c_1(x), x \in (E - cl_1(F_n)) \cap H\}$ minimum de différentiel de coût entre les arcs "potentiels" externes à la fermeture et qui divergent de $H$
    $\delta_{cl2} \leftarrow \min\{c_2(x) - c_2(y), y \in cl_2(F_n) - F_n - H, x \in S_2(x, F_n) \cap F_n \cap H\}$ Le minimum de différentiel de coût entre les arcs "potentiels" internes à la fermeture 2 ; qui divergent de $H$
    $\delta_2 \leftarrow \min\{m_2 - c_2(y), y \in (E - cl_2(F_n)) - H\}$ minimum de différentiel de coût entre entre les arcs "potentiels" externes à la fermeture et qui divergent de $H$
    $\delta \leftarrow \min\{\delta_{H1}, \delta_1, \delta_{H2}, \delta_2\}$
    **for all** $e \in H$ **do**
      $c_1(e) \leftarrow c_1(e) + \delta$
      $c_2(e) \leftarrow c_2(e) - \delta$
    **end for**
  **end if**
**end while**

**Algorithm 6** Ensemble $E^+(e)$ des arêtes prédécesseurs de $e$

**Require:** $G = (V, E)$ $c_k(e) \forall \mathcal{E}_o$
**Ensure:** $E^+(e)$ ensemble des arêtes prédécesseurs de $e$
1: $E^+(e) \leftarrow \emptyset$
2: **for all** $e' \in \mathcal{E}_o \cap F_n$ **do**
3: **if** $c_k(e) = c_k(e')$ **then**
4: $E^+(e) \leftarrow E^+(oj) \cup \{e'\}$
5: **end if**
6: **end for**

**Algorithm 7** Ensemble $E^-(e)$ des arêtes successeurs de $e$

**Require:** $G = (V, E)$ $c_k(e) \forall e \in E$
**Ensure:** $E^-(e)$ ensemble des arêtes successeurs de $e$
1: $E^-(e) \leftarrow \emptyset$
2: **for all** $e'E$ tels que $F_n \cup \{e'\}$ contient un cycle et $F_n - \{e\} \cup \{e'\}$ n'en contient pas (stigme) **do**
3: **if** $c_k(e) = c_k(e')$ **then**
4: $E^-(e) \leftarrow E^-(e) \cup \{e'\}$
5: **end if**
6: **end for**

**Algorithm 8** Arbre couvrant de poids minimum, avec $deg(o) \leq k$

**Require:** $G = (V, E)$ et fonction de coût $c : E \rightarrow \mathbb{R}^+$
**Ensure:** Arbre couvrant de poids minimal, avec $deg(o) \leq k$ (si réalisable)
1: $n \leftarrow 0$, $F_0 \leftarrow \emptyset$, $c_g(e) \leftarrow c(e)$, $c_k(e) \leftarrow 0, \forall e \in E$
2: **while** $E - cl_g(F_n) \neq \emptyset$ et $E - cl_k(F_n) \neq \emptyset$ **do**
3: trouver les éléments $E_g \subset E$ de coût minimums $c_g(e)$ à ajouter à $F_n$ tel que cela ne forme pas de cycle
4: trouver les éléments de $E_k$ qui peuvent être ajoutés : il s'agit soit de $E$ (contrainte de degré non serrée) ou de $E - \mathcal{E}_o$ (contrainte serrée)
5: **if** $E_g \cap E_k \neq \emptyset$ **then**
6: $F_{n+1} \leftarrow F_n \cup \{e^*\}$ pour un $e^* \in E_g \cap E_k$
7: $n \leftarrow n + 1$
8: **else**
9: Construire $G = (E, A)$ pour chaque $e \in \mathcal{E}_o \cap F_n$ à l'aide des algorithmes 7 et 6
10: Trouver l'ensemble $H$ des arêtes atteintes depuis $E_g$
11: **if** $\exists e^* \in H \cap E - CL_k$ par le chemin $p$ **then**
12: $F_n{+}1 \leftarrow F_n \Delta p$ différence symétrique
13: $n \leftarrow n + 1$
14: **else**
15: Calculer $\delta$
16: $c_k(e) \leftarrow c_k(e) - \delta, \forall H$
17: $c_g(e) \leftarrow c_g(e) + \delta, \forall H$
18: **end if**
19: **end if**
20: **end while**

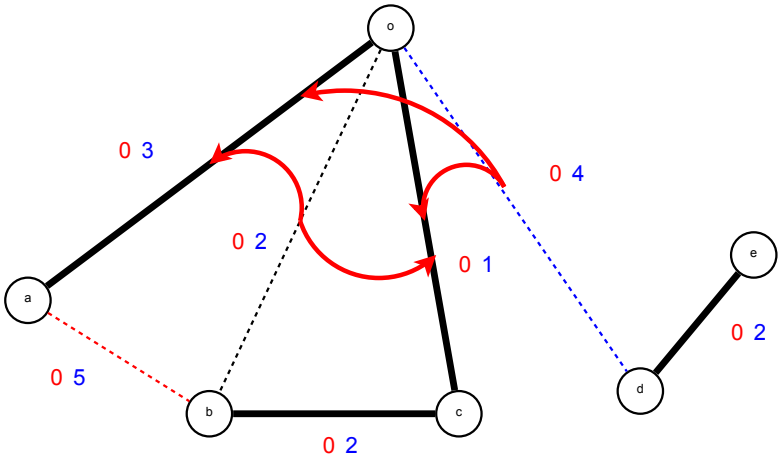



FIGURE 2.7 – Création du graphe $G = (E, A)$.

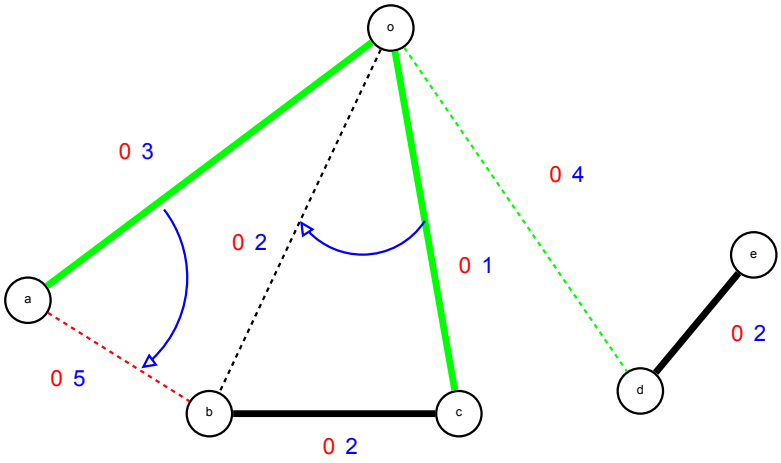



FIGURE 2.8 – Calcul de $\delta$.

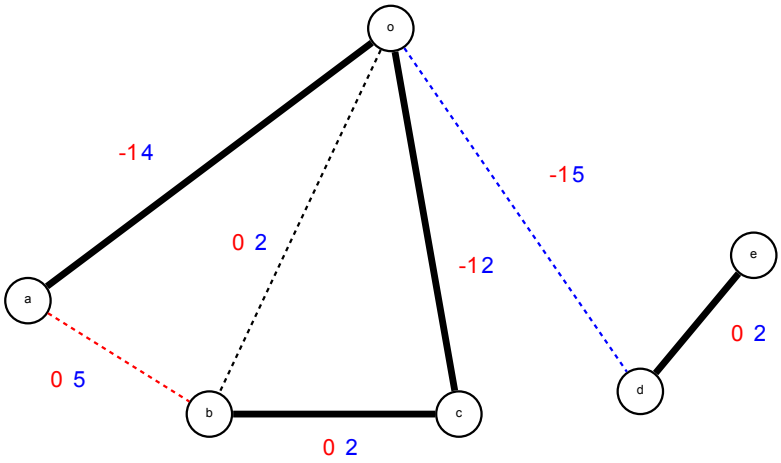



FIGURE 2.9 – Modification des coûts.

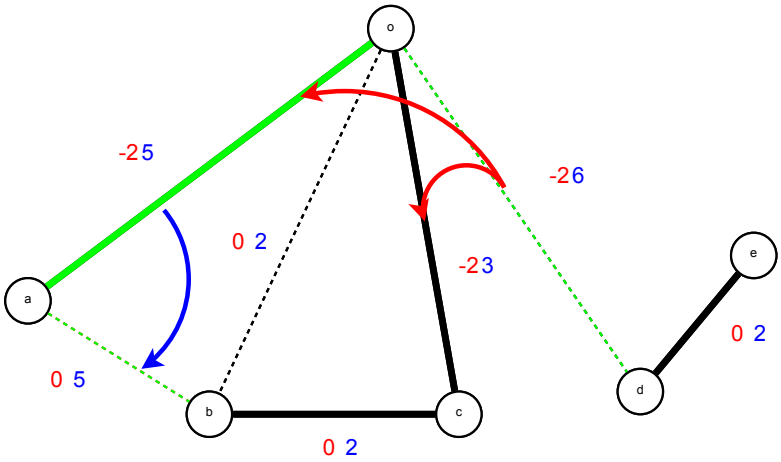



FIGURE 2.10 – Solution optimale.

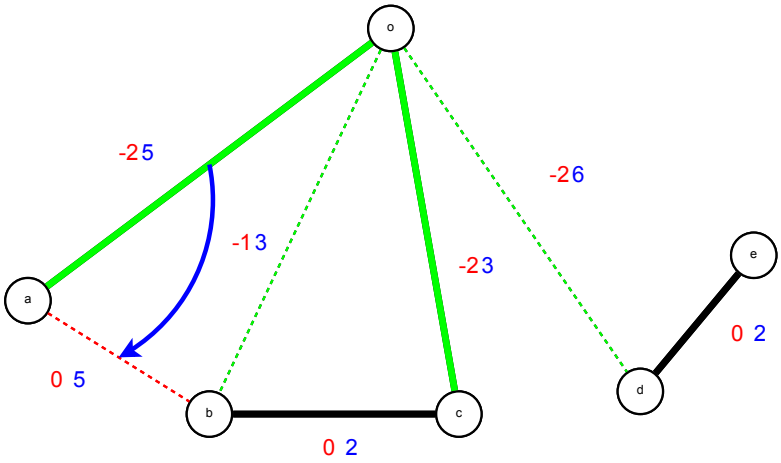



FIGURE 2.11 – Modification des coûts.

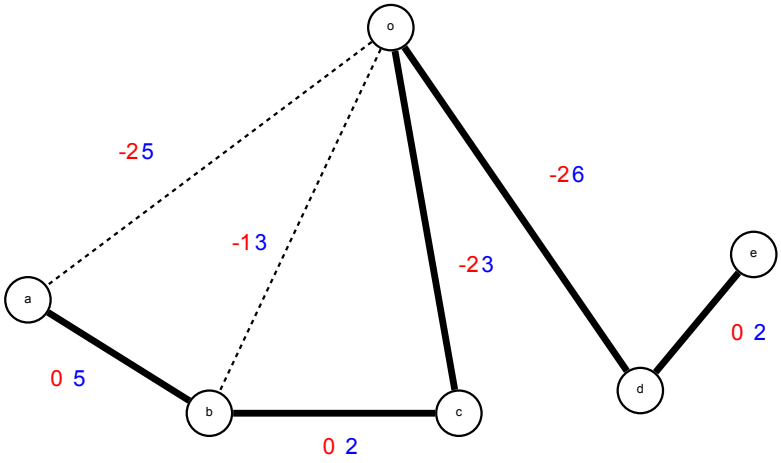



FIGURE 2.12 – Solution optimale.

# Chapitre 3

# Localisation de caches hiérarchiques dans une arborescence, à hit ratio variable

## Introduction

Les programmes dynamiques peuvent permettre de décomposer polynomialement certains problèmes dans les arborescences, tels que la localisation de serveurs et/ou de contenus ([68],[98] et [61]. Cependant ces approches de programmation dynamique polynomiales imposent des hypothèses trop simplificatrices pour notre problème, et qui de surcroît, ne prennent pas en compte le comportement réel d'un cache, notamment au niveau de son contenu. Il est également possible d'étendre ces algorithmes pour résoudre des problèmes plus difficiles au prix d'une complexité plus élevée ([72]). A l'inverse, cette approche plus souple qui permet de prendre en compte d'autres contraintes est, en pratique peu performante. Nous motivons ici l'idée d'appréhender ce paradigme d'une autre façon, qui sera à la fois efficace et facilement adaptable à d'autres problématiques de localisation (principalement dans les arborescences).

L'hypothèse de hit ratio constant dans un problème de localisation de caches est peu réaliste pour deux raisons. La première est qu'en réalité le hit ratio d'un cache dépend principalement de la distribution de la popularité et de la taille de sa mémoire, comme nous le montre expérimentalement [105], dans la figure 3.1

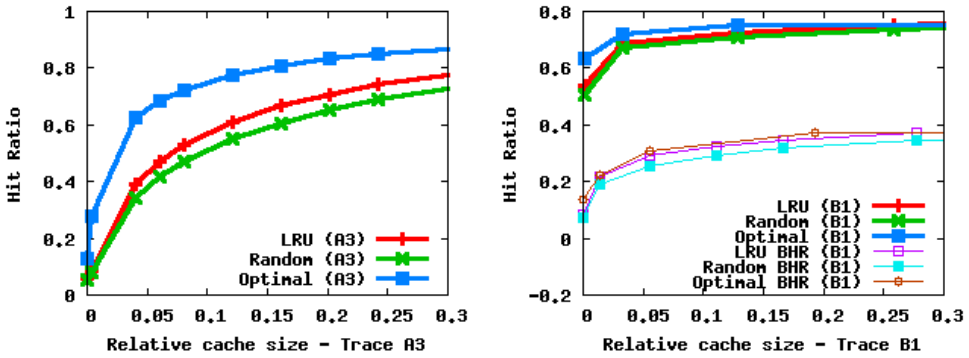



FIGURE 3.1 – *Deux courbes expérimentales donnant le hit ratio, en fonction des politiques de remplacements, de la taille du cache, mesuré sur du trafic de vidéo à la demande (à gauche), et du trafic "Web streaming" (à droite).*

La seconde raison est qu'il est impossible d'estimer le comportement de caches successifs dans une hiérarchie puisque, comme on l'a vu, le modèle nous contraint à séparer les données (et les requêtes correspondantes) "cachables" et non "cachables". L'approche donnée par [72], est centrée sur des classes de contenus. Les limites d'une telle méthode par rapport à notre problème résident tout d'abord dans le mode de transmission. L'hypothèse de "multi-cast" suppose une simultanéité des demandes, ce qui est faux dans notre cas. De plus, il est possible d'optimiser le placement de contenus, qui dans le cas de caches n'est pas un choix, mais un comportement contraint. Nous allons à présent introduire des variables de contenus pour nos caches, ce qui nous permettra de connaitre en chaque nœud sur lequel se trouve un cache, la valeur suivante $\sum_{n \in N_t} x_n.D_{n,t}$. Nous pourrons dès lors connaitre la satisfaction des demandes $D_{n,t}$ si la classe de contenu $n$ est présente, et donc de déduire le hit ratio déjà défini par (définition 3) :

$$HR(t) = \frac{\sum_{n \in N_t} x_n.D_{n,t}}{\sum_{n \in N_t} D_{n,t}}$$

La distribution de popularité par type de contenus est ici une donnée d'entrée du problème, qui s'appuie sur des estimations statistiques. Plus précisément, nous nous concentrerons sur le moment où la demande est la plus forte (le pic). C'est en effet cette référence qui est généralement utilisée pour permettre de construire un réseau ; c'est pourquoi dans le domaine des télécommunications on le caractérise comme "dimensionnant". Nous regrouperons ensuite ces contenus en un nombre $N$ de classes de contenus, suivant un critère de popularité, et en nous assurant que la taille mémoire de chaque classe de contenus est similaire (la somme des volumes des contenus de la classe). Une manière simple de choisir ce nombre $N$ est de faire correspondre la taille de ces groupes avec la capacité mémoire des caches. Ceci est particulièrement intéressant puisque cela permet d'exprimer la capacité d'un cache uniquement par le nombre de classes qu'il peut contenir.

Nous considérerons que la période choisie est représentative du pic, et qu'elle se trouve dans un régime permanent. Cette hypothèse ne prend pas en compte les effets transitoires (mécanismes de remplacement, corrélation temporelles durant la période, bruit, etc...). Cependant le point de

vue macroscopique du modèle (agrégation des contenus par classe, description non nominative des fichiers), est moins sensible et absorbe en partie ces effets locaux.

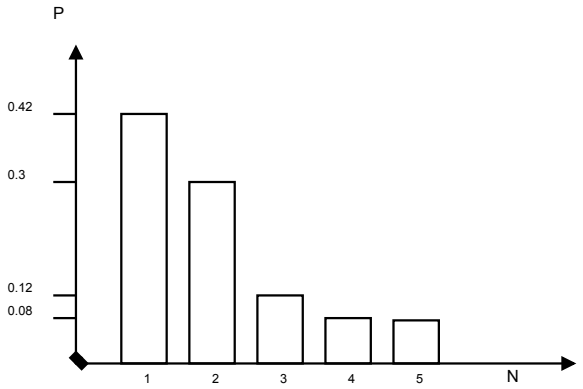



FIGURE 3.2 – *Un exemple de distribution de popularité, que l'on peut assimiler à la distribution des probabilités de demande des classes de fichiers, segmentée en 5 classes de contenus.*

## 3.1 Mesure de la demande et de l'espérance

**Définition 10.** *Nous définissons l'ensemble des choix de classes fichiers possibles* $\Omega = [\![1; N]\!]$. *Soit la variable aléatoire* $X$ *de* $\Omega$ *vers* $\mathbb{R}^N$. *La loi de probabilité de* $X$ *associée est notée* $\mathbb{P}_X$. *Enfin la probabilité qu'un fichier* $n$ *soit objet d'une demande client est notée* $\mathbb{P}_X(\{n\})$ *que nous raccourcirons en* $p_n$.

Nous pourrons alors définir un flux moyen de demandes, résultant de l'espérance de la variable aléatoire $X$.

**Définition 11.** *En appelant la taille* $T_n$ *du fichier* $n$. *Le résultat de l'expérience* $X(\{n\})$ *est le vecteur :*

$$X(\{n\}) = \begin{cases} 0 \ \forall n' \in N \setminus \{n\} \\ T_n \qquad\qquad\qquad\quad \textit{sinon.} \end{cases}$$

Nous pourrons alors définir l'espérance suivante :

**Définition 12.** *L'espérance générale de la variable* $X$ *est définie par le vecteur :*

$$E(X) = X(\{n\}).\mathbb{P}_X(\{n\}) = (T_1.p_1, ..., T_n.p_n)$$

Pour un nœud $i$, nous précisons alors l'espérance en considérant également la population de clients $D_i$ en ce nœud.

**Définition 13.** *En considérant une population de clients* $D_i$ *au nœud* $i$, *l'espérance* $E_i(X)$ *(abrégée en* $E_i$*) de la variable aléatoire de* $X$ *sur tous les clients au nœud* $i$ *est donnée par :*

$$E_i = (E_{i,1} = T_1.p_1.D_i, ..., E_{i,n} = T_n.p_n.D_i)$$

Dans le cadre d'une arborescence, les requêtes non satisfaites par un fils, seront reformulées à son nœud père. Cela signifie que l'on doit distinguer l'espérance de trafic de la population d'un nœud $i$ et l'espérance globale de mesurée sur la sous-arborescence $\mathcal{T}^i$ induite par la nœud $i$ (comprenant donc la racine $i$). L'espérance de flux/demande nous est alors donnée par la relation suivante :

**Proposition 16.** *L'espérance $E_{\mathcal{T}^i,n}$ mesurée sur toute la sous-arborescence induite par $i$ pour le contenu $n$ selon les espérances $E_{\mathcal{T}^j,n}$ des sous-arborescences de ses fils $j \in \delta^+(i)$ est donnée par la relation suivante :*

$$E_{\mathcal{T}^i,n} = \sum_{j \in \delta^+(i)} E_{\tau_j,n} + E(i,n)$$

*Démonstration.* Soient $p_{i,n}$ et les $p_{\mathcal{T}^j,n}, \forall j \in \delta^+(i)$ respectivement les probabilités du nœud $i$ et de toutes les sous-arborescences induites par $j \in \delta^+(i)$ de requérir le contenu $n$. Les espérances sont respectivement $E_{i,n} = T_n.D_i.p_{i,n}$ et $E_{\mathcal{T}^j,n} = T_n.D_{\mathcal{T}^j}.p_{\mathcal{T}^j,n}$. L'espérance est un opérateur linéaire, et les variables étant définies sur le même espace probabiliste, on a :

$$E_{\mathcal{T}^i,n} = T_n.D_i.p_{i,n} + \sum_{j \in \delta^+(i)} T_n.D_{\mathcal{T}^j}.p_{\mathcal{T}^j,n}, \forall n \in N$$

Soit :

$$E_{\mathcal{T}^i,n} = \sum_{j \in \delta^+(i)} E_{\tau_j,n} + E(i,n)$$

□

Plus généralement, si un cache est présent au nœud $i$, et si celui-ci possède la classe de contenus $n$ ($x_{i,n} = 1$), la totalité de cette demande sera absorbée, modifiant donc l'espérance globale sur toute la sous-arborescence de racine $i$.

**Proposition 17.** *Soit l'espérance $E_{\mathcal{T}^i,n}$mesurée sur toute la sous-arborescence induite par $i$ pour le contenu $n$ selon les espérances $E_{\mathcal{T}^j,n}$ des sous-arborescences de ses fils $j \in \delta^+(i)$. En considérant la variable $x_{i,n}$ de placement de la classe de contenu $n$ en $i$,*

$$E_{\mathcal{T}^i,n} = (1 - x_{i,n}).(\sum_{j \in \delta^+(i)} E_{\tau_j,n} + E(i,n))$$

*Démonstration.* Lorsqu'un cache est placé au nœud $i$ et possède le contenu $n$, la probabilité de requête $n$ de toute l'arborescence de racine $i$ tombe à 0, quelque soit la distribution de popularité de $\mathcal{T}^i$. L'espérance est donc nulle également. □

Puisqu'à chaque requête formulée en amont est associé un flot de contenu associé en aval, nous assimilerons dorénavant cette espérance à un flot (ou une espérance de flot). Nous allons à présenter un modèle de localisation de caches transparents.

## 3.2 Modèle de localisation de caches

Nous rappelons à nouveau que $\mathcal{T} = (\mathcal{V}, \mathcal{A})$ désigne le graphe arborescent de racine $o$, le serveur central. Les coûts des arcs incident à $i \in V$ sont notés $w_i$. Par convention $w_o = 0$. Nous ajoutons également une capacité d'arc $c_i, \forall i \in V$, qui représente une limitation de débit sur l'arc incident à $i$. Dans le cadre d'un trafic élastique, c'est-à-dire lorsque les requêtes peuvent être satisfaites plus tard pour cause de congestion, ce terme définit alors une limite de congestion imposée, alors que dans le cas d'un trafic non élastique (ou "circuit"), cette limitation est uniquement le débit supporté par le lien physique. La capacité mémoire d'un cache, modélisée par un nombre de classes, sera notée $C_m$, tandis que son coût sera noté $C_c$. Enfin, pour chaque nœud, le nombre de clients sera noté $D_i$. La probabilité qu'une classe de contenus soit l'objet d'une demande au

nœud $i$ est notée $p_{i,n}, \forall i \in V, \forall n \in N$. La taille d'un fichier moyen $T_f$ sera commune à chaque classe de contenus.

Pour décrire notre modèle nous utilisons les variables suivantes :

1. $y_i$ : la variable de décision binaire de valeur 1 si un cache est installé au nœud $i$, 0 sinon.
2. $x_{i,n}$ : la variable de décision binaire de valeur 1 si le cache installé au nœud $i$ contient la classe de contenus $n$.
3. $f_{i,n}$ : la variable qui représente le flux de contenus incident à $i$. Il est alors de valeur égale à l'espérance $E_{\mathcal{T}^i,n}$ (selon la relation 3.1) de demandes formulées sur toute l'arborescence de racine $i$.
4. $F_{i,n}$ : la variable qui représente la mesure de l'espérance avant interception éventuelle d'un cache

$$
LCA \begin{cases}
\min \sum_{i \in V} C_c.y_i + \sum_{n \in N} \sum_{i \in V} w_i f_{i,n} & & (3.1) \\
\sum_{n \in N} x_{i,n} \leq C_m.y_i, & \forall i \in V & (3.2) \\
f_{i,n} = (1 - x_{i,n}).F_{i,n}, & \forall i \in V, \forall n \in N & (3.3) \\
\sum_{n \in N} f_{i,n} \leq c_i, & \forall i \in V & (3.4) \\
F_{i,n} = \sum_{j \in \delta^+(i)} f_{j,n} + D_i.T_f.p_{i,n}, & \forall i \in V, \forall n \in N & (3.5) \\
x_{i,n}, y_i \in \{0,1\}, f_{i,n} \geq 0, & i \in V, n \in N & (3.6)
\end{cases}
$$

L'objectif (3.1) consiste à minimiser le coût du déploiement de l'architecture, composé à la fois de coût d'installation de caches et de la consommation moyenne de la bande passante sur les liens. La première contrainte (3.2), décrit la limitation mémoire qu'impose la configuration du cache. Puisque les classes ont été construites en fonction de la taille mémoire du cache, celle-ci ne s'exprime plus qu'en nombre de classes. La deuxième contrainte (3.3) définie l'espérance mathématique selon les espérances des nœuds fils comme explicité par la proposition 3.1. La contrainte (3.4) est une limitation $c_i$ en terme de volume de trafic. Enfin la contrainte (3.5) introduit une mesure de l'espérance avant une éventuelle interception de tout dispositif de cache.

Dans ce modèle, les caches se comportent comme des serveurs, car leurs contenus ne sont pas contraints et demeurent donc des variables à optimiser conjointement avec le placement. Nous avons déjà vu que les politiques de remplacement des caches ont toutes pour but de se rapprocher de la politique optimale (MIN/oracle), qui elle-même vise à contenir à chaque moment, les fichiers les plus populaires. En oubliant l'effet transitoire qu'impliquent les algorithmes de remplacement et les perturbations dynamiques, nous ajoutons une hypothèse de régime stationnaire en considérant que le cache contient les fichiers les plus populaires, selon la mesure qui en est faite sur son nœud. Pour la prendre en compte, nous ajoutons donc la contrainte suivante :

$$
(x_{i,n} - x_{i,n'})(F_{i,n} - F_{i,n'}) \geq 0, \ \forall (n, n') \in N^2, \forall i \in V \tag{3.7}
$$

La contrainte se comporte alors comme suit ; si le contenu $n$ est plus populaire que le contenu $n'$ (la relation d'ordre est effectuée sur la mesure de l'espérance mathématique sur la sous-arborescence globale), cela contraindra le contenu $n$ à être automatiquement contenu si le contenu $k'$ l'est aussi. De manière symétrique si le contenu $n$ est présent et le contenu $n'$ ne l'est pas, c'est que la popularité de $n$ soit être supérieur à $n'$. Si les deux contenus $k$ et $k'$ sont présents, alors le contrainte est relâchée. Le modèle complet "Localisation de caches dans une arborescence (LCA)" devient :

$$
LCA \begin{cases}
\min \sum_{i \in V} C_c.y_i + \sum_{n \in N} \sum_{i \in V} w_i f_{i,n} & & (3.8) \\
\sum_{n \in N} x_{i,n} \leq C_m.y_i, & \forall i \in V & (3.9) \\
f_{i,n} = (1 - x_{i,n}).F_{i,n}, & \forall i \in V, \forall n \in N & (3.10) \\
\sum_{n \in N} f_{i,n} \leq c_i, & \forall i \in V & (3.11) \\
(x_{i,n} - x_{i,n'})(F_{i,n} - F_{i,n'}) \geq 0, & \forall (n,n') \in N^2, \forall i \in V & (3.12) \\
F_{i,n} = \sum_{j \in \delta^+(i)} f_{j,n} + D_i.T_f.p_{i,n}, & \forall i \in V, \forall n \in N & (3.13) \\
x_{i,n}, y_i \in \{0,1\}, f_{i,n} \geq 0, & i \in V, n \in N & (3.14)
\end{cases}
$$

Notons que ce modèle n'est pas linéaire (contraintes 3.10 et 3.12 ). Si des méthodes élaborées permettent de résoudre ce type de problème, nous présenterons ici une méthode de linéarisation "classique". On peut effectuer la première linéarisation (3.10) :

$$
(1 - x_{i,n}). \sum_{j \in \mathcal{T}^i} D_j.T_f.p_{j,n} \geq f_{i,n} \geq F_{i,n} - x_{i,n}. \sum_{j \in \mathcal{T}^i} D_j.T_f.p_{j,n}, \forall i \in V, \forall n \in N \quad (3.15)
$$

et

$$
\sum_{j \in \delta^+(i)} f_{j,n} + (1 - x_{i,n}).D_i.T_f.p_{i,n} \geq f_{i,n}, \forall i \in V, \forall n \in N \quad (3.16)
$$

Avec $\mathcal{T}^i \subset V$, les nœuds présents dans tout le sous-arbre de racine $i$.

Si $x_{i,n} = 1$ alors $f_{i,n} = 0$ puisque $f_{i,n} \geq 0$. Si, en revanche, $x_{i,n} = 0$, alors $f_{i,n}$ sera encadrée par les deux équations précédentes imposant donc $f_{i,n} = \sum_{j \in \delta^+(i)} f_{j,n} + D_i.T_f.p_{i,n}$.

Les contraintes (3.12) peuvent, quant à elles, être linéarisées comme suit :

En définissant un "grand M" comme suit :

$$
M_{i,n,n'} = \max_{n,n'} (\sum_{j \in \mathcal{T}^i} D_j.T_f.p_{j,n}) \quad (3.17)
$$

$$
F_{i,n} - F_{i,n'} \geq (x_{i,n} - x_{i,n'} - 1).M_{i,n,n'}, \forall (n,n') \in N^2, \forall i \in V \quad (3.18)
$$

$$
LCA \begin{cases}
\min \sum_{i \in V} C_c.y_i + \sum_{n \in N} \sum_{i \in V} w_i f_{i,n} & & (3.19) \\
\sum_{n \in N} x_{i,n} \leq C_m.y_i, & \forall i \in V & (3.20) \\
f_{i,n} \geq F_{i,n} - x_{i,n}. \sum_{j \in \mathcal{T}^i} D_j.T_f.p_{j,n}, & \forall i \in V, \forall n \in N & (3.21) \\
f_{i,n} \leq F_{i,n}, & \forall i \in V, \forall n \in N & (3.22) \\
\sum_{n \in N} f_{i,n} \leq c_i, & \forall i \in V & (3.23) \\
F_{i,n} - F_{i,n'} \geq (x_{i,n} - x_{i,n'} - 1).M_{i,n,n'}, & \forall (n,n') \in N^2, \forall i \in V & (3.24) \\
F_{i,n} = \sum_{j \in \delta^+(i)} f_{j,n} + D_i.T_f.p_{i,n}, & \forall i \in V, \forall n \in N & (3.25) \\
x_{i,n}, y_i \in \{0,1\}, f_{i,n} \geq 0, & i \in V, n \in N & (3.26)
\end{cases}
$$

## 3.3 Définitions, notations et propriétés

Il est intéressant d'observer que la règle de préférence (contraintes (3.24)) ne constitue pas une inégalité valide du problème de localisation de serveurs et de contenus. Ainsi le comportement qui induit de contenir et d'intercepter la classe de fichiers la plus populaire ne constitue pas nécessairement la stratégie optimale en terme de coût de la fonction objectif. Pour s'en convaincre, il suffit d'observer la figure 3.3.

En supposant que la taille mémoire du cache n'est capable de contenir qu'une seule classe de contenu, et que son coût est nul : nous exhibons sur la figure 3.3 la solution optimale en relâchant la contrainte de préférence. Le coût la de solution est donné par la somme des flux moyens sur chaque lien multiplié par le coût linéaire de transmission correspondant. Nous obtenons donc des flux $f_{b,1} = (1 - x_{b,1}).p_{b,1}.D_b = 0.66$ et $f_{b,2} = (1 - x_{b,1})p_{b,2}.D_b = 0$ de coûts de transmission unitaire égal à 1, et des flux $f_{a,1} = (1 - x_{a,1}).(p_{a,1}.D_a + f_{b,1}) = 0$ et $f_{a,2} = (1 - x_{a,2}).(p_{a,2}.D_a + f_{b,2}) = 9 \times 0.33 = 2.97$ avec un coût de transmission linéaire égal à 10 : la somme nous donne un coût de 30.36.

Au contraire nous exhibons sur la figure 3.4 la solution optimale avec la contrainte (3.24) —contrainte de préférence. Le coût de la solution est donné par la somme des flux moyens sur chaque lien multiplié par le coût linéaire de transmission correspondant. Nous obtenons donc des flux $f_{b,1} = (1 - x_{b,1}).p_{b,1}.D_b = 0$ et $f_{b,2} = (1 - x_{b,1})p_{b,2}.D_b = 0.33$ de coûts de transmission unitaire égal à 1, et des flux $f_{a,1} = (1-x_{a,1}).(p_{a,1}.D_a+f_{b,1}) = 0$ et $f_{a,2} = (1-x_{a,2}).(p_{a,2}.D_a+f_{b,2}) = 9 \times 0.33 + 0.33 = 3.33$ avec un coût de transmission linéaire égal à 10 : la somme nous donne un coût de 33.63. La différence majeure avec la solution sans contrainte de préférence vient du fait qu'au nœud a, le coût linéaire de transmission est critique, il est donc préférable de minimiser le flux total à cet endroit, et c'est la configuration précédente qui le permet.

Nous allons introduire quelques définitions supplémentaire :

**Définition 14.** *Nous appelons $y_{\mathcal{T}^i} \in \{0,1\}^{|\mathcal{T}^i|}$ le vecteur de localisation de caches dans la sous-arborescence $\mathcal{T}^i$, de racine $i$ — nœud $i$ compris. Par extension $y_i$ est alors la valeur particulière*

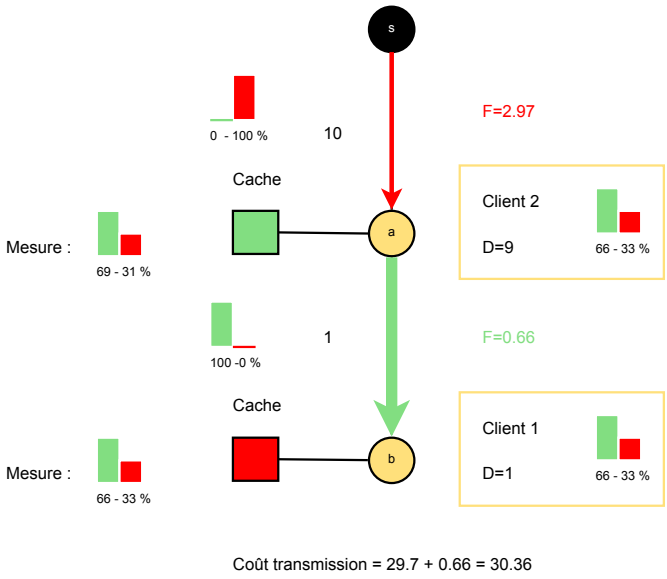



FIGURE 3.3 – *Stratégie optimale de placement de contenus.*

*en $i$ du même vecteur. L'ordre utilisé est un ordre lexicographique : un nœud de profondeur supérieure aura toujours un indice de valeur supérieure.*

**Définition 15.** *Nous appelons $Y_{\mathcal{T}^i}$ l'ensemble des vecteurs de localisation de cache du sous-arbre $\mathcal{T}^i$ de racine $i$. Il est défini comme :*

$$Y_{\mathcal{T}^i} \in \{0,1\}^{\mathcal{T}^i} \tag{3.27}$$

**Définition 16.** *Soit $I(F,n,C_m)$ la fonction qui vaut 1 si la classe de contenus $n$ est parmi les $C_m$ éléments les plus populaires, selon une configuration de flux $F$, et 0 sinon.*

**Définition 17.** *Nous utiliserons, $f_{i,y_{\mathcal{T}^i}}$ : le vecteur qui représente l'espérance de flux $f_{i,n,y_{\mathcal{T}^i}}$ pour chacune des classes $n$ en chaque nœud $i$, sachant les décisions $y_{\mathcal{T}^i}$ prises dans le sous-arbre $\mathcal{T}^i$. Il est notamment défini par les relations suivantes :*

$$F_{i,y_{\mathcal{T}^i}} = \sum_{j \in \delta^+(i)} f_{j,y_{\mathcal{T}^j}} + p_i.D_i.T_f \tag{3.28}$$

*et $F_{i,y_{\mathcal{T}^i}}$ avec :*

$$f_{i,n,y_{\mathcal{T}^i}} = (1 - I(F_{i,y_{\mathcal{T}^i}}, n, y_i.C_m)).(F_{i,n,y_{\mathcal{T}^i}}), \forall n \in N \tag{3.29}$$

**Définition 18.** *Nous définissons à présent $REAL_{\mathcal{T}^i}$, l'ensemble des vecteurs $Y_{\mathcal{T}^i}$ réalisables, définit par :*

$$REAL_{\mathcal{T}^i} = \{y_{\mathcal{T}^i} \in \{0,1\}^{\mathcal{T}^i}, ||f_{i,y_{\mathcal{T}^i}}||_1 \leq c_i\} \tag{3.30}$$

**Définition 19.** *Soient deux vecteurs $f_{i,y_{\mathcal{T}^i}}$ et $f_{i,y'_{\mathcal{T}^i}}$. Nous définissons la relation d'ordre :*

$$f_{i,y_{\mathcal{T}^i}} \succcurlyeq f_{i,y'_{\mathcal{T}^i}} \Leftrightarrow f_{i,n,y_{\mathcal{T}^i}} \geq f_{i,n,y'_{\mathcal{T}^i}}, \forall n \in N \tag{3.31}$$

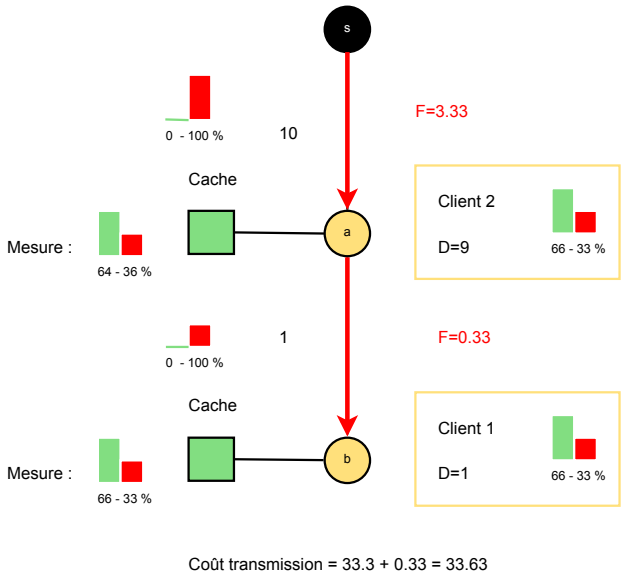



FIGURE 3.4 – *Comportement réel des caches.*

**Définition 20.** *Nous définissons le coût $C(y_{\mathcal{T}^i})$ d'une solution par :*

$$C(y_{\mathcal{T}^i}) = \sum_{j \in \mathcal{T}^i} y_j.C_c + ||f_{j,y_{\mathcal{T}^j}}||_1.w_j \tag{3.32}$$

**Définition 21.** *Soit un vecteur $y^+_{\mathcal{T}^i}$. Nous définissons le sous-problème dit "de domination", suivant :*

$$y^*(y^+_{\mathcal{T}^i}) = \arg \min_{\substack{y_{\mathcal{T}^i} \in REAL_{\mathcal{T}\rangle} \\ f_{i,y^+_{\mathcal{T}^i}} \succcurlyeq f_{i,y_{\mathcal{T}^i}}}} C(y_{\mathcal{T}^i}) \tag{3.33}$$

Ce problème de domination vise à connaître la solution la moins chère sous contraintes de flux total maximum, donné par la solution associée au vecteur $y^+_{\mathcal{T}^i}$. Le résultat de ce problème nous donnera la solution de coût minimal selon ces contraintes.

## 3.4 Programme dynamique, localisation de caches au hit ratio variable dans un réseau arborescent

Le modèle linéaire présenté précédemment nécessite beaucoup de contraintes et de variables, et les relaxations offrent des bornes assez faibles en pratique. Nous allons donc nous intéresser à la spécificité de notre problématique, à savoir que notre graphe est une arborescence. Nous allons alors tirer parti de cette caractéristique en réutilisant des concepts de récursion déjà présentés précédemment, et en les appliquant à notre problème.

Dans [68], [98] et [61], où il s'agit de placer des caches capable d'intercepter totalement un flux connu, et dont les liens ne possèdent pas de capacités, le principe de récursion permet, étant donnée une décision prise *a priori* (à savoir, placer ou non un cache au nœud considéré), de

créer une somme de sous-problèmes devenus indépendants. Appliquer ici ce principe est impossible pour deux raisons majeures.

Tout d'abord il y a les contraintes (3.23) du modèle explicité précédemment imposent une réalisabilité de capacité au nœud $i$. Si l'on peut contraindre la solution à être réalisable (par la mise en place d'un cache par exemple), on ne garantit plus l'optimalité de la solution retournée, comme nous le montre l'exemple de la figure 3.5. La solution (solution irréalisable) qu'un programme dynamique de type [98] destiné au placement de caches à l'interception de classe fixé et sans capacité (en fonction de coût uniquement ou k-localisation), pourrait proposer (Solution "réparée"). Sans limitation sur le nombre de caches à installer (k-médian), quelle que soit la décision prise au nœud "routeur", il n'est pas intéressant d'un point de vue économique de placer un cache aux nœuds clients. Une fonction récursive ne permettra donc pas de placer un cache à ces nœuds. Pour satisfaire la contrainte de capacité, le programme devra installer un cache. Pourtant la solution optimale respectant la contrainte de capacité nécessite d'installer un cache en un nœud feuille (Solution optimale).

De plus les contraintes (3.24), nous obligent à avoir une mesure de la popularité à chaque nœud $i$, elle même dépendante de ce qui a été installé dans le sous arbre $\mathcal{T}^i$. Cela signifie que l'information sur le placement de contenus est nécessaire pour prendre la décision de mettre ou non un cache.

Luss ([72]) évite cet écueil en calculant chaque état réalisable du nœud courant (au sens du nœud père) que devra satisfaire la solution locale de la sous-arborescence. Dans la problématique qu'il aborde, si le nombre de ces états reste limité et constant par nœud, l'approche ne semble pas suffisamment efficace. Dans notre problème le nombre de ces états (les flux de demandes possibles) est extrêmement élevé, et dépend de la profondeur du nœud considéré dans l'arborescence.

La récursion de Luss s'appuie également peu sur la dominance des coûts, et la décomposabilité du problème. En effet, des états sont souvent plus souples que d'autres. Si la solution locale optimale sur la sous-arborescence $\mathcal{T}^i$ supposait par exemple l'état le plus optimiste $e_i$ (comme des demandes remontées nulles), alors il était en réalité inutile de tester les autres états. La structure même de son algorithme de programmation dynamique ne permet pas d'établir cette connaissance *a priori*. Cependant le choix de l'ordre de ces états (notamment du plus "optimiste" au plus "pessimiste") pourrait réduire cette recherche.

Nous allons donc proposer une nouvelle méthode pour pouvoir capter ce phénomène et s'appuyer sur des règles de dominance pour supprimer le maximum d'états et s'appuyer sur l'indépendance qui peut survenir sur une sous-arborescence optimale (une sous-arborescence ne dépendant plus de son père).

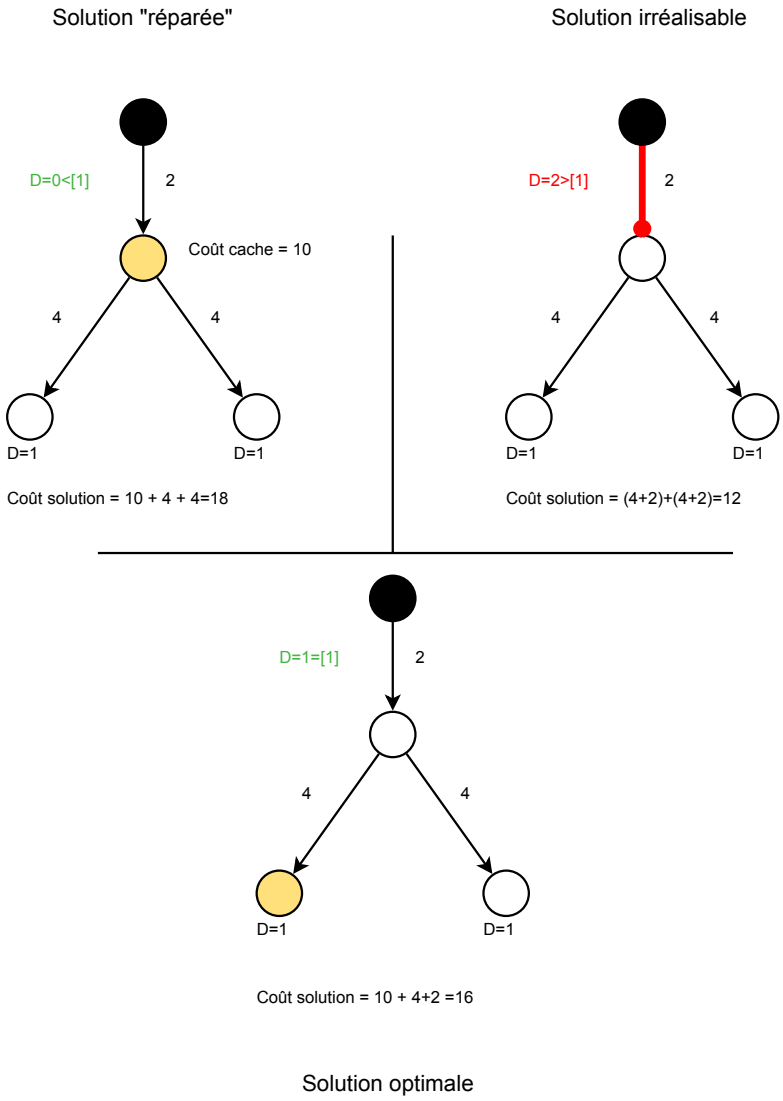



FIGURE 3.5 –

La solution que nous proposons consiste à recréer un espace de solutions sans présupposer une connaissance sur le nœud en cours, comme il est généralement fait dans les programmes dynamiques. Nous allons simplement construire récursivement l'espace des solutions possibles, comme suit :

**Définition 22.** *Nous appelons $Y^*(\mathcal{T}^i)$ l'espace des solutions dominantes, construit de la façon suivante :*

$$Y^*(\mathcal{T}^i) = \{y^*(y^+_{\mathcal{T}^i}), y^+_{\mathcal{T}^i} \in Y(\mathcal{T}^i)\} \tag{3.34}$$

**Définition 23.** *$Y(\mathcal{T}^i)$, l'espace des solutions courantes est défini par :*

$$Y(\mathcal{T}^i) = \{y_{\mathcal{T}^i} \in \{0,1\} \times \prod_{j \in \delta^+(i)} Y^*(\mathcal{T}^j)\} \tag{3.35}$$

Cette relation de récurrence est alors utilisée dans l'algorithme dont le pseudo-code est résumé sur 9. Nous récupérons les ensembles des solutions dominantes fils $j$ de $\delta^+(i)$ (étapes 1 à 3). Nous créons les nouvelles solutions possibles, à savoir le produit cartésien des solutions au nœud $i$ et tous ses fils (étapes 4 à 6). Si ce nœud ne possède pas de fils, il s'agit simplement de placer un cache ou non (étapes 6 à 8). Il s'agit ensuite de supprimer les solutions dominées deux à deux par flots et coûts comparés (étapes 9 à 16). Nous pouvons retourner l'ensemble ainsi construit.

**Algorithm 9** Solutions dominantes $Y^*(\mathcal{T}^i)$

**Require:** $\mathcal{T} = G(V, A)$ arborescence de racine $i$, $C_c$ coût d'un cache
**Ensure:** Ensemble Solutions dominantes $Y^*(\mathcal{T}^i)$
1: **for all** $j \in \delta^+(i)$ **do**
2: Calcul de $Y^*(\mathcal{T}^j)$
3: **end for**
4: **if** $\delta^+(i) \neq \emptyset$ **then**
5: $Y(\mathcal{T}^i) \leftarrow \{0,1\} \times \prod_{j \in \delta^+(i)} Y^*(\mathcal{T}^j)$ déterminer le nouvel ensemble des possibles (3.35)
6: **else**
7: $Y(\mathcal{T}^i) \leftarrow \{0,1\}$
8: **end if**
9: **for all** $y^i_{\mathcal{T}} \in Y(\mathcal{T}^i)$ **do**
10: **for all** $y'_{\mathcal{T}^i} \in Y(\mathcal{T}^i) - y_{\mathcal{T}^i}$ **do**
11: **if** $f_{i,y'_{\mathcal{T}^i}} \succcurlyeq f_{i,y_{\mathcal{T}^i}}$ et $C(y'_{\mathcal{T}^i}) \geq C(y_{\mathcal{T}^i})$ **then**
12: $Y(\mathcal{T}^i) \leftarrow Y(\mathcal{T}^i) - y'_{\mathcal{T}^i}$ (3.34)
13: **end if**
14: **end for**
15: **end for**
16: $Y^*(\mathcal{T}^i) \leftarrow Y(\mathcal{T}^i)$
17: **return** $Y^*(\mathcal{T}^i)$

Nous allons à présent détailler le fonctionnement de l'algorithme appliqué à l'exemple décrit dans la figure 3.6.

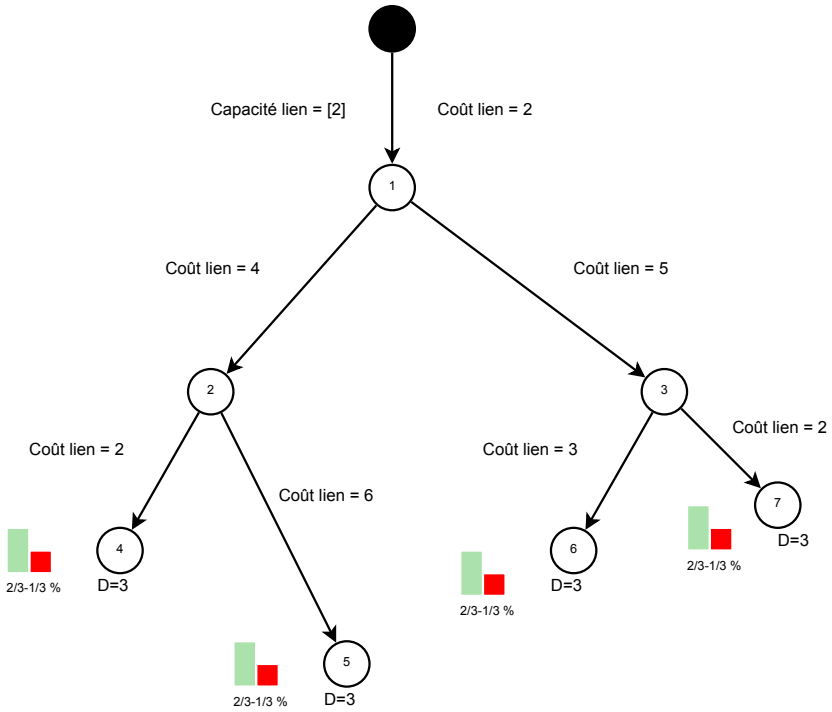



FIGURE 3.6 – *L'arborescence constituée du serveur central (en noir), les coûts des liens $c_i$, les demandes en chaque nœud ainsi que la popularité des deux classes de contenus (en vert et rouge), avec le volume global de requêtes $D_i = D = 3, \forall i \in V$. Un cache de coût $C_c = 18$ ne peut contenir qu'une classe de contenus et peut être installé sur chacun des nœuds.*

Nous commençons par calculer les ensembles dominants sur les feuilles $Y^*(\mathcal{T}^4), Y^*(\mathcal{T}^5), Y^*(\mathcal{T}^6)$ et $Y^*(\mathcal{T}^7)$. Les sous-solutions sont données par la figure 3.7. Aucune solution ne domine l'autre (si une solution est moins chère, le vecteur de flux n'est pas comparable à l'autre du point de vue de la relation d'ordre de la définition 3.3. Nous nous concentrons alors à présent sur le nœud 2. La figure 3.8 résume d'abord l'ensemble des solutions possibles $Y(\mathcal{T}^2)$, puis l'ensemble des solutions dominantes $Y^*(\mathcal{T}^2)$. Le tableau 3.1 résume également les ensembles $Y(\mathcal{T}^3)$ et $Y^*(\mathcal{T}^3)$. Enfin nous décrivons $Y(\mathcal{T}^1)$ et $Y^*(\mathcal{T}^1)$. Puisque le lien incident en 1 ne supporte pas un flux global de 2, certaines solutions sont immédiatement rejetées. Puisque nous sommes à la racine de notre arborescence, nous pouvons alors immédiatement choisir la solution qui coûte le moins cher, donnée ici par le vecteur $(0, 1, 1, 1, 1, 0, 0)*$, et représentée par la figure 3.9.

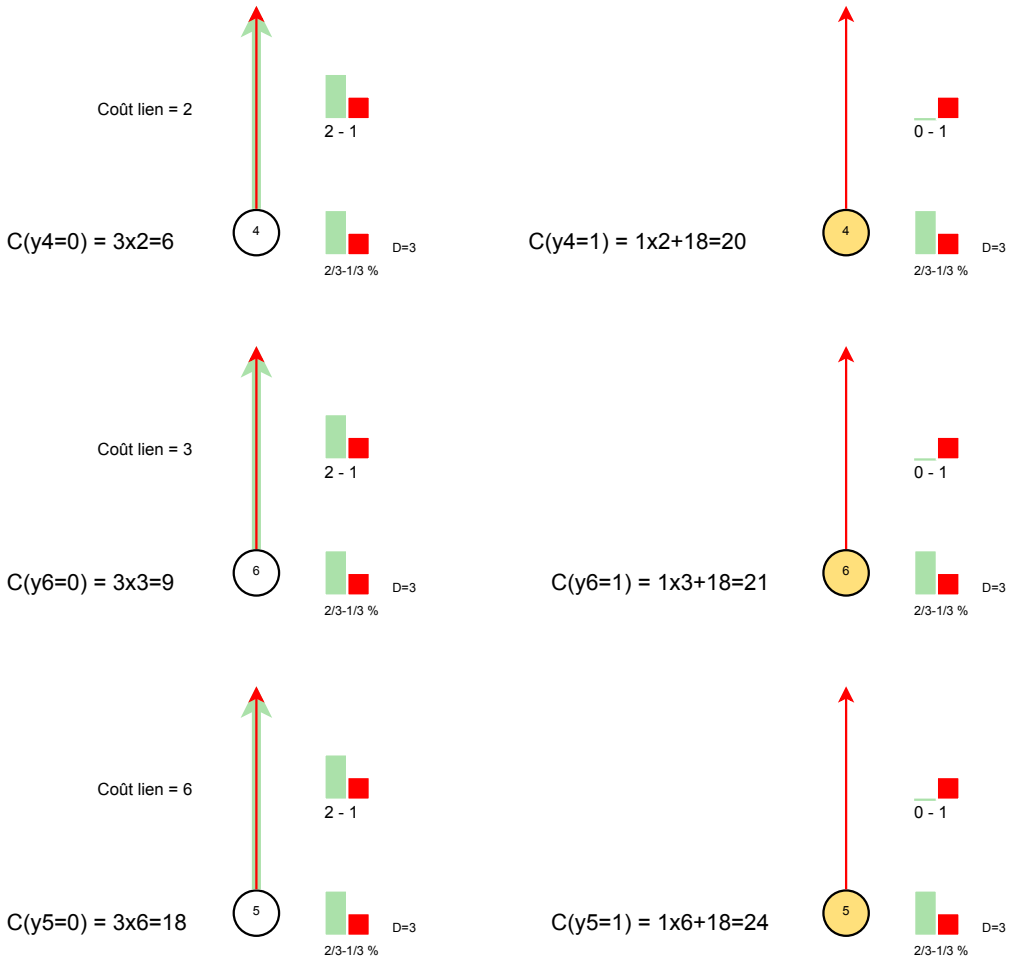



FIGURE 3.7 – *Pour chaque nœud feuille, la solution sans cache (à gauche), et avec un cache (à droite). Le coût associé à la solution et le vecteur de demandes pour chaque classe de contenus sont également indiqués. Le nœud 7 n'est pas représenté ici, car il est identique au nœud 4.*

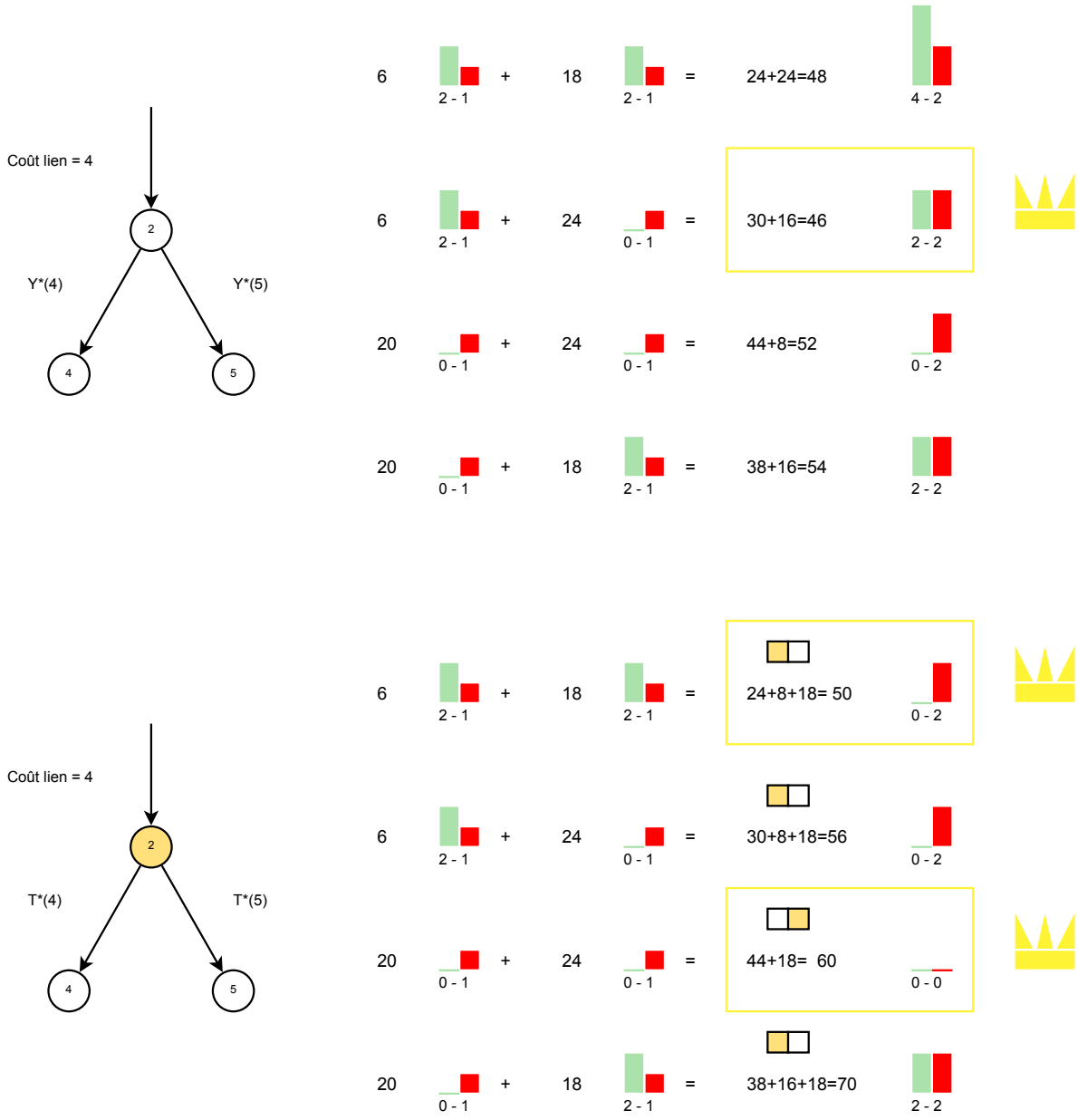



FIGURE 3.8 – *Calcul de $Y(\mathcal{T}^2)$ en effectuant le produit cartésien des solutions des fils (4 et 5), et de la solution associée à la décision de placer ou non un cache au nœud courant 2. Il suffit ensuite de récupérer les solutions dominantes (annotées d'une couronne) pour constituer $Y^*(\mathcal{T}^2)$.*

| $C(\mathcal{T}^3)$ | | |
|---|---|---|
| $y_{\mathcal{T}^3} = (y_3, y_6, y_7)$ | $f_{3,y_{\mathcal{T}^3}}$ | $C(y_{\mathcal{T}^3})$ |
| (0,0,0) | (4,2) | $C((0)^6) + C((0)^7) + f_{3,y_{\mathcal{T}^3}}.w_3 = 9 + 6 + 6.5 = 45$ |
| (0,1,0) | (2,2) | $C((1)^6) + C((0)^7) + f_{3,y_{\mathcal{T}^3}}.w_3 = 21 + 6 + 4.5 = 47$ |
| (0,0,1) | (2,2) | $C((0)^6) + C((1)^7) + f_{3,y_{\mathcal{T}^3}}.w_3 = 9 + 20 + 4.5 = 49$ |
| (0,1,1) | (0,2) | $C((1)^6) + C((1)^7) + f_{3,y_{\mathcal{T}^3}}.w_3 = 21 + 20 + 2.5 = 51$ |
| (1,0,0)* | (0,2) | $C((0)^6) + C((0)^7) + f_{3,y_{\mathcal{T}^3}}.w_3 + C_c = 9 + 6 + 2.5 + 18 = 43$ |
| (1,1,0) | (0,2) | $C((1)^6) + C((0)^7) + f_{3,y_{\mathcal{T}^3}}.w_3 + C_C = 21 + 6 + 2.5 + 18 = 55$ |
| (1,0,1) | (0,2) | $C((0)^6) + C((1)^7) + f_{3,y_{\mathcal{T}^3}}.w_3 + C_c = 9 + 20 + 2.5 + 18 = 57$ |
| (1,1,1)* | (0,0) | $C((1)^6) + C((1)^7) + f_{3,y_{\mathcal{T}^3}}.w_3 + C_c = 21 + 20 + 18 = 59$ |

TABLE 3.1 – Déclinaison de l'ensemble $Y(\mathcal{T}^3)$, et extraction de son ensemble $Y^*(\mathcal{T}^3)$, annoté par '$*$'.

| $C(\mathcal{T}^1)$ | | |
|---|---|---|
| $y_{\mathcal{T}^1} = (y_1, y_2, ..., y_7)$ | $f_{1,y_{\mathcal{T}^1}}$ | $C(y_{\mathcal{T}^1})$ |
| $(0,0,0,1,1,0,0)^{\times}$ | (2,4) | $C((0,0,1)^2) + C((1,0,0)^3) + f_{y_{\mathcal{T}^1}}.w_1$ |
| $(0,0,0,1,1,1,1)^{\times}$ | (2,2) | $C((0,0,1)^2) + C((1,1,1)^3) + f_{y_{\mathcal{T}^1}}.w_1$ |
| $(0,1,0,0,1,0,0)^{*\times}$ | (0,4) | $C((1,0,0)^2) + C((1,0,0)^3) + f_{y_{\mathcal{T}^1}}.w_1$ |
| (0,1,0,0,1,1,1) | (0,2) | $C((1,0,0)^2) + C((1,1,1)^3) + f_{y_{\mathcal{T}^1}}.w_1$ |
| (0,1,1,1,1,0,0)* | (0,2) | $C((1,1,1)^2) + C((1,0,0)^3) + f_{y_{\mathcal{T}^1}}.w_1$ |
| (0,1,1,1,1,1,1) | (0,0) | $C((1,1,1)^2) + C((1,1,1)^3) + f_{y_{\mathcal{T}^1}}.w_1$ |
| (1,0,0,1,1,0,0) | (2,0) | $C((0,0,1)^2) + C((1,0,0)^3) + f_{y_{\mathcal{T}^1}}.w_1 + C_c$ |
| (1,0,0,1,1,1,1) | (0,2) | $C((0,0,1)^2) + C((1,1,1)^3) + f_{y_{\mathcal{T}^1}}.w_1 + C_c$ |
| (1,1,0,0,1,0,0)* | (0,0) | $C((1,0,0)^2) + C((1,0,0)^3) + f_{y_{\mathcal{T}^1}}.w_1 + C_c$ |
| (1,1,0,0,1,1,1) | (0,0) | $C((1,0,0)^2) + C((1,1,1)^3) + f_{y_{\mathcal{T}^1}}.w_1 + C_c$ |
| (1,1,1,1,1,0,0) | (0,0) | $C((1,1,1)^2) + C((1,0,0)^3) + f_{y_{\mathcal{T}^1}}.w_1 + C_c$ |
| (1,1,1,1,1,1,1) | (0,0) | $C((1,1,1)^2) + C((1,1,1)^3) + f_{y_{\mathcal{T}^1}}.w_1 + C_c$ |

| $C(\mathcal{T}^1)$ | | |
|---|---|---|
| $y_{\mathcal{T}^1} = (y_1, y_2, ..., y_7)$ | $f_{1,y_{\mathcal{T}^1}}$ | $C(y_{\mathcal{T}^1})$ |
| $(0,0,0,1,1,0,0)^{\times}$ | (2,4) | $46 + 43 + 6.2 = 101$ |
| $(0,0,0,1,1,1,1)^{\times}$ | (2,2) | $46 + 59 + 4.2 = 113$ |
| $(0,1,0,0,1,0,0)^{*\times}$ | (0,4) | $50 + 43 + 4.2 = 101$ |
| (0,1,0,0,1,1,1) | (0,2) | $50 + 59 + 2.2 = 113$ |
| (0,1,1,1,1,0,0)* | (0,2) | $60 + 43 + 2.2 = 107$ |
| (0,1,1,1,1,1,1) | (0,0) | $60 + 59 = 119$ |
| (1,0,0,1,1,0,0) | (2,0) | $46 + 43 + 2.2 + 18 = 111$ |
| (1,0,0,1,1,1,1) | (0,2) | $46 + 59 + 2.2 + 18 = 127$ |
| (1,1,0,0,1,0,0)* | (0,0) | $50 + 43 + 18 = 111$ |
| (1,1,0,0,1,1,1) | (0,0) | $50 + 59 + 18 = 127$ |
| (1,1,1,1,1,0,0) | (0,0) | $60 + 43 + 18 = 121$ |
| (1,1,1,1,1,1,1) | (0,0) | $60 + 59 + 18 = 137$ |

TABLE 3.2 – Déclinaison de l'ensemble $Y(\mathcal{T}^1)$, et extraction de son ensemble $Y^*(\mathcal{T}^1)$, annoté par '$*$'. Les solutions qui ne sont pas réalisables sont notées '$\times$'.

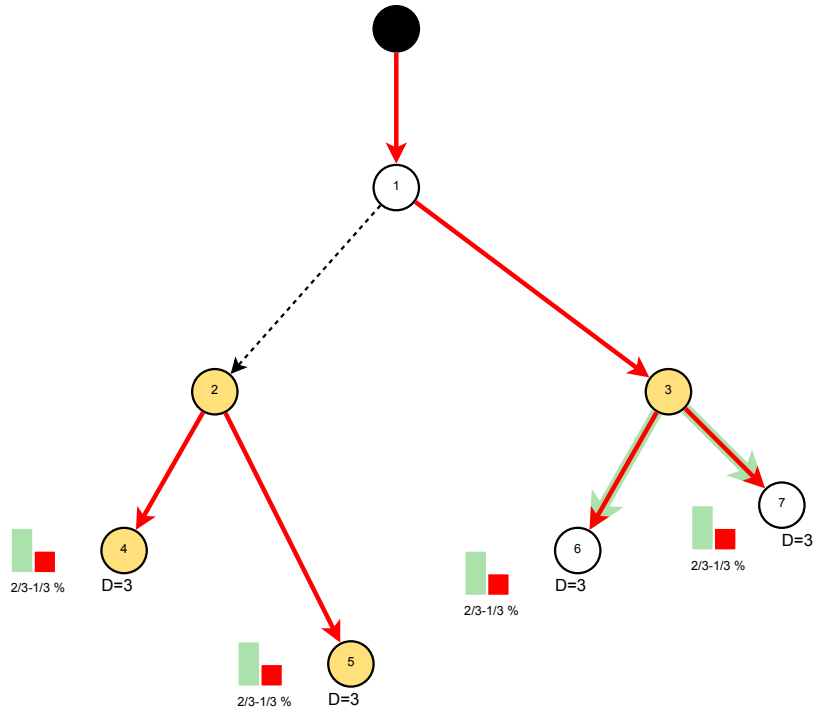



FIGURE 3.9 – *Solution optimale, de coût 107 (se référer aux tables).*

Une amélioration très classique pour résoudre des instances plus facilement et limiter l'explosion combinatoire, est de déformer l'arborescence, de sorte qu'un nœud n'ait au maximum que deux fils, comme le montre l'exemple de la figure 3.10. Cette déformation implique que le nœud $t$ ne doit pas être éligible pour l'installation d'un cache.

En pratique, la mise en place d'une telle procédure pourrait être faite durant la création des produits cartésiens, par décomposition en sous-groupes.

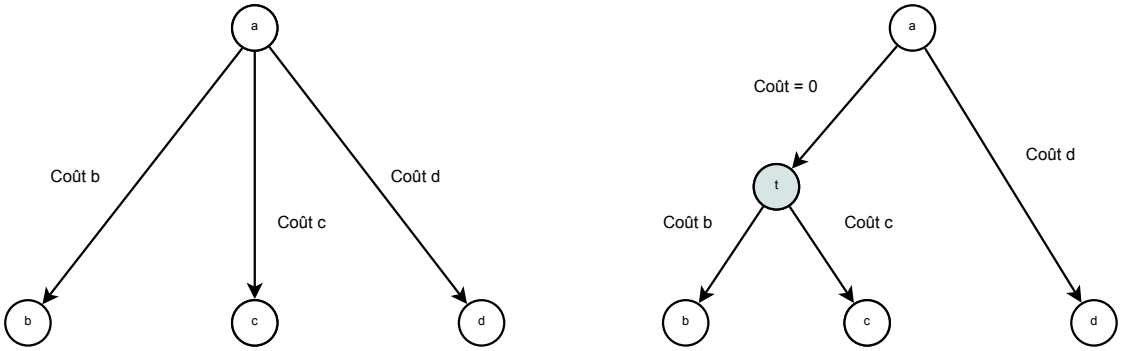



FIGURE 3.10 – *A gauche l'arborescence initiale, à droite, la version avec deux fils.*

## 3.5 Expérimentations

Pour réaliser les expérimentations, nous avons utilisé un générateur d'arborescences, dont les paramètres sont donnés ci-dessous :

1. Population de clients choisie uniformément dans l'intervalle $[\![1, 5]\!]$
2. Taille de fichier unitaire
3. Capacité de l'arc choisi uniformément dans l'intervalle : [10,50]
4. Coût de transmission linéaire de l'arc choisi uniformément dans l'intervalle :[1,20]
5. $|N| = 5$ classes de contenus
6. Distribution de popularité $[0.33, 0.27, 0.2, 0.13, 0.07]$ (de type Zipf)
7. Capacité de caches $C_m = 2$
8. Coût cache $C_m = 2$
9. Clients localisés aux feuilles

Ces instances sont construites de manière à obtenir des solutions optimales non triviales.

| $\|V\|$ | PLNE | DP | Ecart (%) |
|---|---|---|---|
| 210 | 4 | 3 | 0,017 |
| 255 | 431 | 3 | 0 |
| 322 | 23 | 4 | 0 |
| 375 | 52 | 3 | 0 |
| 393 | 73 | 4 | 0 |
| 455 | 338 | 2 | 0 |
| 538 | >3600 | 5 | 0,018 |
| 1431 | >3600 | 45 | -0,13 |

TABLE 3.3 – PLNE et programme dynamique. Le temps de calcul a été limité à une heure. Les écarts constatés sont reportés dans la dernière colonnes. Un écart négatif signifie que la programmation dynamique a trouvé une solution meilleure que la meilleure solution donnée par la PLNE au bout de la limite de temps. L'ordinateur utilisé est doté d'un processeur Intel Core 2 Duo P8700 de fréquence 2.53 GHz, et de 3.48 Go de RAM. Le solveur utilisé est Xpress (Mosel 3.20 - mmsystem 1.8.8 - mmxprs 2.2.0).

La complexité de cet algorithme n'est pas bornée polynomialement : elle est donc potentiellement exponentielle. Pourtant, en pratique, beaucoup de solutions sont dominées (comme nous avons pu le voir dans l'exemple 3.6) et peuvent être écartées "à la volée" ; ce sont des solutions dont le coût est facilement estimé et comparé, et qui ne seront donc même pas construites. En pratique nous limitons l'utilisation de la mémoire avec ce procédé, une grande quantité de mémoire ainsi qu'un grand nombre de calcul, ce qui n'était pas le cas pour [72]. Ceci rend notre algorithme particulièrement efficace, même sur de grandes instances. La figure 3.11 nous montre un exemple de dominances qui surviennent naturellement au cours de l'algorithme.

La règle de dominance 3.3 peut également supprimer une sous-solution locale qui constituerait la solution optimale. L'exemple 3.12 nous l'illustre. La solution (à gauche) avec une solution dominée rejetée, qui constitue pourtant partie de la solution optimale (à droite). Les demandes/flux qui s'ajoutent aux nœuds intermédiaires, sont des solutions sur les sous-arborescences (non représentées) équivalentes dans les deux solutions. Les mesures des flux avant éventuelle interception

par un cache sont données sur les nœuds tandis que les mesures de flux après éventuelle interception sont notées sur les arcs. Il est important de noter que cet exemple a pour but de montrer l'abus créé par la règle de dominance ; contraindre ici le PLNE à intercepter exactement ce qu'il peut contenir (passer les contraintes (3.20) à l'égalité) rendrait alors la solution de droite impossible. Fort heureusement, la perte d'optimalité constatée en pratique lors des expérimentations reste anecdotique.

Les résultats obtenus montrent d'abord l'écart constaté par rapport à l'optimalité. En pratique cet écart est extrêmement faible (au plus ici de 0.018%). L'algorithme de programmation dynamique permet en outre de résoudre des tailles d'instances relativement élevées.

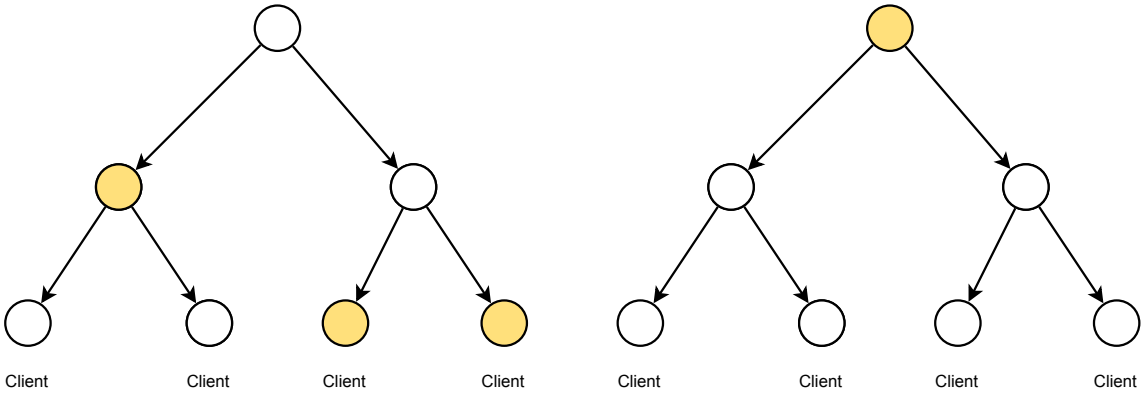



FIGURE 3.11 – *Dans le cas où les clients sont aux feuilles ; ces deux solutions sont nécessairement comparables avec la relation d'ordre déjà citée, puisque le flux entrant à la racine est le même mais le nombre de caches diffère.*

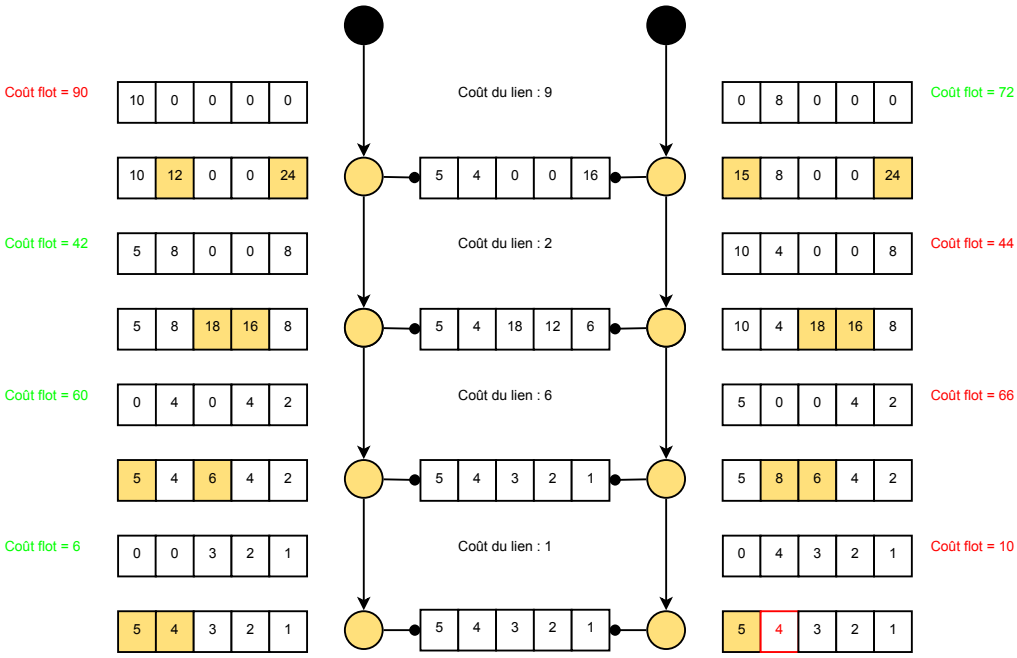



FIGURE 3.12 – *Exemple de non optimalité.*

L'intérêt d'une telle méthode réside dans sa capacité à s'étendre sur d'autres problèmes de localisation dans des graphes arborescents. Nous allons maintenant présenter des extensions de l'algorithme.

## 3.6 Extension 1 : Localisation de serveurs et de contenus

Si nous relâchons la contrainte de préférence, le cache transparent devient un serveur classique, comme nous l'avons déjà vu. Un grand nombre de choix de contenus devient alors possible. Si nous segmentons les fichiers en classes de fichiers de sorte qu'un cache/serveur puisse contenir exactement une classe de contenus, comme nous le suggérions déjà, les combinaisons possibles deviennent égales au nombre de ces classes.

Nous appellerons $x_{i,n}$ la variable en $\{0,1\}$ précisant quelle classe de contenus est choisie pour prendre place dans le serveur (le cas échéant).

Dans ce cas, la relation de dominance devient exacte.

**Proposition 18.** *Soient $i$ et $j$ deux nœuds tels que $i \in \mathcal{T}^j$. Si $f_{i,y_{\mathcal{T}^i}} \succcurlyeq f_{i,y'_{\mathcal{T}^i}}$ et $C(y_{\mathcal{T}^i}) > C(y'_{\mathcal{T}^i})$. Alors la solution optimale en $y^*_{\mathcal{T}^j}$ ne contient pas $y_{\mathcal{T}^i}$.*

*Démonstration.* Supposons que la solution optimale en $j$ soit uniquement constituée de $y_{\mathcal{T}^i}$. Les autres $y_k, k \in \mathcal{T}^j - \mathcal{T}^i$, restent inchangés. Nous avons la relation suivante :

$$f_{i,n,y_{\mathcal{T}^i}} = (1 - x_{i,n}).(\sum_{j \in \delta^+(i)} f_{j,n,y_{\mathcal{T}^j}} + p_{i,n}.D_i.T_f) \tag{3.36}$$

Dans le sous arbre $\mathcal{T}^i$, nous avons un flot $f_{i,y_{\mathcal{T}^i}} \succcurlyeq f_{i,y'_{\mathcal{T}^i}}$. D'après la relation (17), sachant que les $y_k, k \in \mathcal{T}^j - \mathcal{T}^i$ restent inchangés nous pouvons alors déduire que :

$$f_{l,y_{\mathcal{T}^l}} \succcurlyeq f_{l,y'_{\mathcal{T}^l}}, \forall l \in \mathcal{T}^j - \mathcal{T}^i$$

Puisque les décisions d'installation sont les mêmes, seuls compte les coûts de transmission. Nous pouvons alors également déduire :

$$C(y_{\mathcal{T}^l}) \geq C(y'_{\mathcal{T}^l}), \forall l \in \mathcal{T}^j - \mathcal{T}^i$$

La solution $y'_{\mathcal{T}^j}$ est donc meilleure ou égale que $y_{\mathcal{T}^j}$, ce qui contredit l'hypothèse.

□

Malgré une génération plus grande de sous-solutions, le programme dynamique reste efficace comme le montre le tableau 3.6, notamment sur des instances de taille élevée.

| $\|V\|$ | DP | PLNE |
|---|---|---|
| 60 | 5s | 3s |
| 85 | 7s | <1s |
| 166 | 18s | 6s |
| 228 | 58s | 59s |
| 288 | 86s | >3600 (0.2%) |
| 183 | 17s | 75s |
| 247 | 25s | 65s |
| 395 | 173s | >3600 (0.3%) |
| 508 | 62s | 1484s |
| 610 | 200s | >3600s (0.4%) |
| 1076 | 423s | >3600s (1.5%) |

TABLE 3.4 – *Comparaison sur instances aléatoires, avec 3 classes de contenus, coût de cache $C_c = 50$, des capacités choisies uniformément dans l'intervalle $[1, 10]$ des coûts choisis uniformément dans l'intervalle $[5, 25]$, et une population en chaque nœud aléatoire uniformément choisie dans l'intervalle $[1, 10]$. L'ordinateur utilisé est doté d'un processeur Intel Core 2 Duo P8700 de fréquence 2.53 GHz, et de 3.48 Go de RAM. Le solveur utilisé est Xpress (Mosel 3.20 - mmsystem 1.8.8 - mmxprs 2.2.0.)*

Si pour des instances de petite taille, le programme linéaire en nombres entiers possède des performances légèrement supérieures, on remarque rapidement un écart sensible entre les temps de résolution au profit de l'approche dynamique. L'augmentation des tailles d'instance est en effet plus favorable au programme dynamique qui observe un différentiel de temps de résolution beaucoup moins drastique. On remarquera malgré tout que les meilleures solutions proposées par le PLNE restent satisfaisantes, avec des écarts d'optimalité (consignés entre parenthèses) assez faibles. Cependant, cette qualité se détériore lorsque l'on cherche à résoudre des instances de grande taille. Dans le même temps, le programme est capable de fournir une solution optimale en un temps raisonnable.

## 3.7 Extension 2 : Caches avec contraintes de capacités

De manière plus générale, un cache peut avoir une capacité de traitement limitée ; c'est à dire que le nombre de requêtes clients/sessions simultanées qu'il peut satisfaire est contraint par la puissance de calcul de son processeur central. Ainsi, dans la pratique, il se peut qu'un cache ne puisse pas délivrer un des contenus qu'il stocke parce que son processeur est surchargé. Ceci va bien évidemment modifier le calcul du hit ratio, ainsi que les popularités mesurée en amont d'un cache surchargé.

Nous appelons capacité de streaming cette capacité notée $C_s$. De plus, nous allons autoriser l'empilement de caches, c'est à dire que les capacités ne seront plus binaires mais entières. Pour ce faire, nous allons émettre l'hypothèse que les caches installés sont coopératifs et opèrent comme une seule entité. On peut dès lors agréger leurs capacités.

Nous ajoutons alors la variable $z_{i,n} \geq 0, \forall i \in V, \forall n \in N$, décrivant l'interception (streaming) effective du cache (ou du groupe de caches) pour la classe de contenus $n$. C'est-à-dire les données que le cache devra être capable de traiter.

$$
LCA \begin{cases}
\min \sum_{i \in V} C_c.y_i + \sum_{n \in N} \sum_{i \in V} w_i f_{i,n} & & (3.37) \\
\sum_{n \in N} x_{i,n} \leq C_m.y_i, & \forall i \in V & (3.38) \\
f_{i,n} = F_{i,n} - z_{i,n}, & \forall i \in V, \forall n \in N & (3.39) \\
\sum_{n \in N} z_{i,n} \leq y_i.C_s, & \forall i \in V & (3.40) \\
z_{i,n} \leq x_{i,n}. \sum_{j \in \mathcal{T}^i} D_j.T_f.p_{j,n}, & \forall i \in V, \forall n \in N & (3.41) \\
z_{i,n} \leq x_{i,n}.D_i.T_f.p_{i,n} + \sum_{j \in \delta^+(i)} f_{j,n}, & \forall i \in V, \forall n \in N & (3.42) \\
F_{i,n} - F_{i,n'} \geq (x_{i,n} - x_{i,n'} - 1).M_{i,n,n'}, & \forall (n,n') \in N^2, \forall i \in V & (3.43) \\
F_{i,n} = \sum_{j \in \delta^+(i)} f_{j,n} + D_i.T_f.p_{i,n}, & \forall i \in V, \forall n \in N & (3.44) \\
x_{i,n}, y_i \in \mathbf{N}, f_{i,n} \geq 0, z_{i,n} \geq 0, & i \in V, n \in N & (3.45)
\end{cases}
$$

Les contraintes (3.39) nous donnent la relation exacte entre les requête associées $n$ connaissant la valeur interceptée $z_{i,n}$ pour celles-ci. Les contraintes (3.40) empêchent la configuration de caches de répondre simultanément à un volume de requêtes supérieur à $y_i.C_s$. Enfin, l'interception d'un type de requêtes ne pourra s'effectuer que si l'agrégation de cache contient la classe de contenus considérée (3.41). Il faudra bien évidemment s'assurer que la nouvelle variable introduite soit supérieure à 0.

Plusieurs requêtes de classes de contenus différentes qui sont bien contenus dans la configuration de caches peuvent apparaître ; mais leur nombre intercepté peut être limité par la capacité de streaming. Le cache ne pourra donc pas toujours permettre de les satisfaire complètement, malgré le fait qu'il les contienne. Indirectement, l'ajout de cette contrainte de capacité introduit une décision quant à la distribution des interceptions pour les classes de contenus présentes dans l'agrégation de caches. Les possibilités d'interception peuvent donc devenir illimitées, et dès lors

empêchent a priori l'utilisation de l'approche dynamique.

Cependant, en pratique, le cache acceptera uniquement les premières requêtes qu'il sera capable de traiter. Il s'agit donc, encore une fois, d'un comportement contraint par le trafic lui-même, et plus exactement l'ordre des requêtes. Les études de la littérature sur le comportement des caches [43] utilisent généralement des modèles d'arrivées de requêtes indépendantes. Cette approximation est appuyée par le fait que nous considérons des classes macroscopiques de contenus, peu sensibles aux variations ponctuelles des contenus. Une première approximation consisterait alors à supposer que les premières requêtes que pourra satisfaire l'agrégation de caches suivra une distribution égale à la distribution de popularité. Ceci nous imposerait l'établissement de la contrainte non-linéaire suivante :

$$\frac{z_{i,n'}}{\sum\limits_{n\in N} z_{i,n}} = \frac{x_{i,n'}.F_{i,n'}}{\sum\limits_{n\in N} x_{i,n}.F_{i,n}}, \forall i \in V, \forall n' \in N \tag{3.46}$$

Le membre de droite de (3.46) est une mesure de la distribution de popularité effectuées sur les classes de contenus uniquement présents dans le caches (dont la présence est statuée par la variable $x_{i,n}$). De cette mesure sera extraite la distribution que devra suivre l'interception décrite par les variables $z_{i,n}$.

Nous reformulons les contraintes (3.46).

$$z_{i,n'} = \min\{\frac{x_{i,n'}.F_{i,n'}}{\sum\limits_{n\in N} x_{i,n}.F_{i,n}}.y_i.C_s, x_{i,n'}.F_{i,n'}\} \forall i \in V, \forall n' \in N \tag{3.47}$$

L'exemple de la figure 3.13 nous montre l'effet de cette contrainte : le cache possède une capacité de 6, ce qui l'empêche d'intercepter la totalité des requêtes de valeur 12, malgré le fait qu'il contienne la totalité des fichiers associés à cette demande. Les proportions d'interceptions sont $[\frac{1}{2}, \frac{1}{3}, \frac{1}{6}]$ sont données par la distribution des requêtes. Le vecteur d'interception est donc donné par $z_i = [6.\frac{1}{2}, 6.\frac{1}{3}, 6.\frac{1}{6}]$. Les requêtes non interceptées (le flux de demande sortant) sont données en amont du cache.

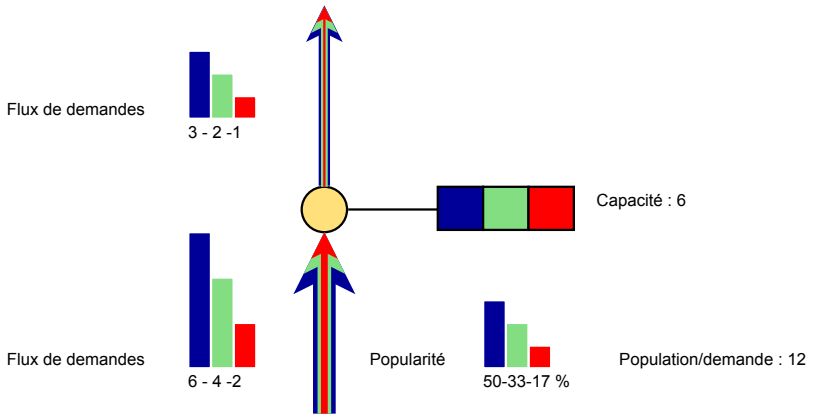



FIGURE 3.13 –

On peut remarquer que l'équation (3.7) de répartition n'intervient réellement que lorsque

$$\frac{y_i.C_s}{\sum\limits_{n\in N} x_{i,n}.(f_{i,n} + D_i.T_f.p_{i,n})} < 1$$

c'est-à-dire lorsque la capacité de streaming $y_i.C_s$ de l'agrégation de caches en $i$ n'est pas suffisante pour satisfaire la totalité des requêtes. Si cette contrainte est difficile à ajouter dans un programme linéaire en nombres entiers, elle est, dans notre approche de programmation dynamique, très facile à intégrer ayant connaissance préalable de la valeur de toutes nos variables du membre de droite. Nous pouvons mesurer l'impact d'une telle contrainte et les performances de l'approche dynamique généralisée, par la tableau 3.5. On remarque alors un écart de valeur de solution pour les petites instances, imputé principalement par la contrainte de distribution. Pourtant, à partir d'une taille d'instance d'environ 200 nœud, le PLNE fournit uniquement une solution approchée au terme de la limite d'un quart d'heure, qui n'est pas même meilleure que celle du programme dynamique, qui satisfait pourtant la contrainte de distribution (par construction).

**Définition 24.** *Nous appelons $M_{F_{i,y_{\mathcal{T}^i}}}$, le nombre maximum de caches à installer pour intercepter la totalité des requêtes du vecteur $F_{i,y_{\mathcal{T}^i}}$ en $i$. Cela signifie que l'agrégation de caches ne doit être limitante, ni au niveau de la capacité de streaming $y_i.C_s$, ni au niveau de la capacité de stockage $y_i.C_m$.*

$$M_{F_{i,y_{\mathcal{T}^i}}} = \max\{\lceil\frac{||F_{i,y_{\mathcal{T}^i}}||_1}{C_s}\rceil, \lceil\frac{\sum\limits_{n\in N} 1^{F_{i,n,y_{\mathcal{T}^i}}>0}}{C_m}\rceil\} \tag{3.48}$$

Nous devrons donc maintenir un plus large ensemble de solutions, comme nous pouvons l'observer dans l'algorithme 10.

Le vecteur d'interception sera à présent défini par $I^2(F, C_m, C_s)$, contraint à la fois par la mémoire ($C_m$) et la capacité de streaming ($C_s$).

**Définition 25.** *Soit $I_2(F, C_m, C_s)$ la fonction d'interception d'un vecteur de flux $F$, sachant une capacité mémoire $C_m$ et une capacité de streaming $C_s$. Elle est définie par :*

$$I_2(F, C_m, C_s)_{n'} = \min\{\frac{I(F, n', C_m).F_{n'}}{\sum\limits_{n\in N} I(F, n, C_m).F_n}.C_s, I(F, n', C_m).F_{n'}\} \tag{3.49}$$

Les flots nous sont donnés, à partir d'une configuration $y_{\mathcal{T}^i} \in \mathbb{N}^{|\mathcal{T}^\rangle|}$, par la relation suivante :

**Définition 26.** *Nous utiliserons, $f_{i,y_{\mathcal{T}^i}}$ : le vecteur qui représente l'espérance de flux $f_{i,n,y_{\mathcal{T}^i}}$ pour chacune des classes $n$ en chaque nœud $i$, sachant les décisions $y_{\mathcal{T}^i}$ prises dans le sous-arbre $\mathcal{T}^i$. Il est notamment défini par les relations suivantes :*

$$F_{i,y_{\mathcal{T}^i}} = \sum_{j\in\delta^+(i)} f_{j,y_{\mathcal{T}^j}} + p_i.D_i.T_f \tag{3.50}$$

*et $f_{i,y_{\mathcal{T}^i}}$ avec :*

$$f_{i,y_{\mathcal{T}^i}} = F_{i,y_{\mathcal{T}^i}} - I_2(F_{i,y_{\mathcal{T}^i}}, y_i.C_m, y_i.C_s) \tag{3.51}$$

Nous pourrons alors reprendre l'algorithme 9 et ajouter la modification suivante : pour chaque configuration possible (produit cartésien des sous-solutions effectué à l'étape 5) nous essayons les solutions de placement de 0 à $M$ caches, $M$ correspondant au nombre minimal de caches pour intercepter la totalité des requêtes (étape 6). Le reste de l'algorithme fonctionne comme 9 ; c'est-à-dire que l'on évacue les solutions dominées selon une règle identique (étapes 12 à 13).

**Algorithm 10** Solutions dominantes $Y^*(\mathcal{T}^i)$

**Require:** $\mathcal{T} = G(V, A)$ arborescence de racine $i$, $C_c$ coût d'un cache
**Ensure:** Ensemble des solutions dominantes $Y^*(\mathcal{T}^i)$

1: **for all** $j \in \delta^+(i)$ **do**
2: Calcul de $Y^*(\mathcal{T}^j)$ (3.34)
3: **end for**
4: **if** $\delta^+(i) \neq \emptyset$ **then**
5: $\tilde{Y}(\mathcal{T}^i) \leftarrow \{0\} \times \prod_{j \in \delta^+(i)} Y^*(\mathcal{T}^j)$ déterminer les combinaisons possibles **sans cache**
6: $Y(\mathcal{T}^i) \leftarrow \sum_{y_{\mathcal{T}^i} \in \tilde{Y}(\mathcal{T}^i)} \left( \sum_{n \in \{0,\dots,M_{\mathcal{F}_{i,y_{\mathcal{T}^i}}}\}} y_{\mathcal{T}^i, y_i = n} \right)$ déterminer l'ensemble des solutions possibles, en mettant au maximum un nombre de caches adapté à l'espérance de requêtes associé
7: **else**
8: $Y(\mathcal{T}^i) \leftarrow \{0, 1, ..., M_{f_{i,y_i=0}}\}$
9: **end if**
10: **for all** $y^i_{\mathcal{T}} \in C(\mathcal{T}^i)$ **do**
11: **for all** $y'_{\mathcal{T}^i} \in C(\mathcal{T}^i) - y_{\mathcal{T}^i}$ **do**
12: **if** $f_{i,y'_{\mathcal{T}^i}} \succcurlyeq f_{i,y_{\mathcal{T}^i}}$ et $C(y'_{\mathcal{T}^i}) \geq C(y_{\mathcal{T}^i})$ **then**
13: $Y(\mathcal{T}^i) \leftarrow Y(\mathcal{T}^i) - y'_{\mathcal{T}^i}$ (3.35)
14: **end if**
15: **end for**
16: **end for**
17: $Y^*(\mathcal{T}^i) \leftarrow Y(\mathcal{T}^i)$
18: **return** $Y^*(\mathcal{T}^i)$

## 3.8 Localisation de deux types de caches, applications numériques

Nous allons nous concentrer ici à la fois sur l'impact de l'introduction d'un deuxième type d'équipement et sur l'importance de leurs caractéristiques. Pour ce faire, nous allons définir plusieurs scénarios contrastés, en considérant leurs données associées les plus réalistes possibles.

Nous définissons tout d'abord un scénario de référence, supposé le plus proche de la réalité.

### 3.8.1 Scénario et cache de référence

Nous considérons à nouveau des instances de graphes arborescents dont la profondeur maximale est de 4 nœuds, et le nombre de fils par nœud non feuille est aléatoirement choisi dans l'ensemble $\{2, 3, 4\}$. Les liens, quant à eux, ont un coût aléatoire pris dans l'intervalle $[5, 20]$ k€, et ont une capacité infinie, afin de pouvoir concentrer l'étude sur les caractéristiques des caches.

| \|V\| | temps PLNE | temps PD | val PLNE | val PD | écart |
|---|---|---|---|---|---|
| 7 | 0,7 | 0,1 | 95.5 | 95.5 | 0 % |
| 27 | 1,1 | 0,1 | 322.2 | 322.2 | 0 % |
| 47 | 12 | 0,1 | 479 | 482 | 0.63 % |
| 85 | 28 | 0,3 | 989 | 999 | 1.01 % |
| 200 | >1000 | 25 | **2300*** | 2306 | - |
| 262 | >1000 | 4.8 | 3252* | **3200** | - |
| 435 | >1000 | 30 | 5213* | **5131** | - |
| 731 | >1000 | 2.4 | 9014* | **8846** | - |
| 953 | >1000 | 36 | 15830* | **11509** | - |

TABLE 3.5 – Comparaison entre l'algorithme dynamique et le programme linéaire (contrainte de distribution relachée). Un astérisque signifie que l'algorithme a été stoppé et la solution fournie est la meilleure courante. L'ordinateur utilisé est doté d'un processeur "Intel Core 2 Duo P8700" de fréquence 2.53 GHz, et de 3.48 Go de RAM. Le solveur utilisé est Xpress (Mosel 3.20 - mmsystem 1.8.8 - mmxprs 2.2.0).

Les caractéristiques du cache de référence sont données dans le tableau 3.8.1.

| **Cache de référence (R) coûts et capacités** |
|---|
| $\forall i \in \mathcal{C}, cost(i) = 20$ k€ |
| $Stockage : C_1 = 2$ block |
| $Streaming : C_2 = 2$ Gbps |

Nous considérerons également deux types de trafic : la vidéo à la demande ("VoD" pour "Video-On-Demand") et les contenus vidéos générés par les utilisateurs (généralement appelés "UGC" pour "User-Generated-Content") dont les caractéristiques sont empruntés à [18, 105] et consignés dans le tableau 3.6.

| param. | **VoD** | **UGC** |
|---|---|---|
| Taille de fichier | 100 MB | 10 MB |
| Taille du catalogue | 100 TB | 1 PB |
| Distribution de popularité | I-Weibull [1] (0.627,0.291) | Zipf(0.961) |
| Débit/session | 2.3 Mbps | 1 Mbps |

TABLE 3.6 – Paramètres des deux trafics considérés

Nous agrégerons les contenus en 10 classes approximativement de même volume (cf. tableaux 3.7 et 3.14 ). Si la taille de la librairie de l'UGC est beaucoup plus grande, nous nous limiterons aux 100 premiers téraoctets, et tronquerons donc la "queue de distribution" de popularité qui a une moindre importance sur l'architecture de déploiement. Les demandes probabilistes de chaque classe en chaque nœud client seront alors données par la relation suivante :

$$p_i^k = \pi^k \times \text{rand}(1000, 10000) \times 10\% \times \text{ bitrate Mbps}, \tag{3.52}$$

où le deuxième terme désigne la population qui se trouve derrière le nœud considéré ; supposant que 10% est le pourcentage de cette population qui requiert réellement les contenus concernés à l'heure de pointe. Enfin le "bitrate" est le flux généré par chaque demande de contenu (i.e. le flux induit par session).

| Block # | $k =$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Popul. | $\pi^k =$ | 60.74 | 15.14 | 7.95 | 4.97 | 3.39 |
| Block # | $k =$ | 6 | 7 | 8 | 9 | 10 |
| Popul. | $\pi^k =$ | 2.46 | 1.85 | 1.44 | 1.14 | 0.92 |

TABLE 3.7 – Courbe typique de popularité VoD discrétisée en 10 groupes.

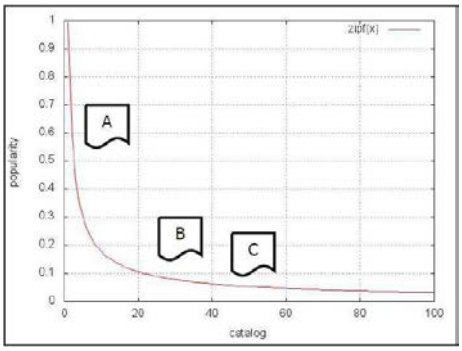



FIGURE 3.14 – *Courbe de popularité standard, discrétisée en blocs.*

Nous utilisons notre modèle pour déterminer le placement optimal de caches sur une moyenne d'une centaine de topologies arborescentes, en faisant varier à chaque fois le coût du cache de référence autour de sa valeur nominale de 20 k€. La figure 3.15 nous montre à la fois le nombre total de caches installés, le coût total du déploiement et enfin, le détail entre les coûts spécifiques d'installations de caches et les coûts de transmission. Nous pouvons observer que lorsque le coût unitaire de cache augmente, le coût global semble augmenter également de façon linéaire. Dans le même temps le nombre de caches effectivement installés diminue, faisant basculer le facteur de coût du coté de la transmission.

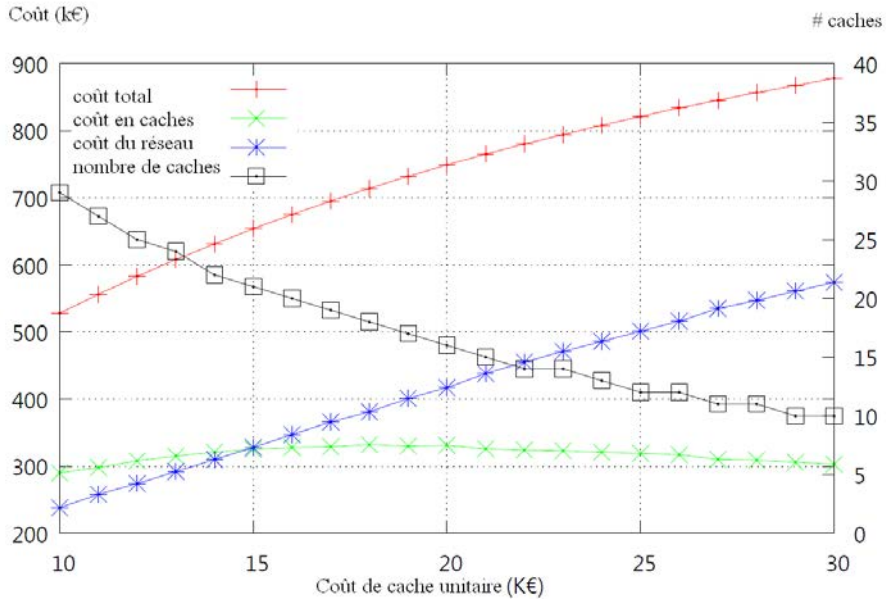



FIGURE 3.15 – *Coûts des solutions optimales (axe à gauche) et nombres de caches installées dans celles-ci (axe à droite), lorsque le coût unitaire de cache varie de 10 à 30 k€(valeur de référence à 20 k€).*

Si nous nous intéressons à présent aux paramètres de capacités (de stockage (3.16) et de streaming (3.17)) du cache de référence, nous remarquons d'abord que l'augmentation de la taille mémoire renforce l'intérêt des caches, et par conséquent plus de caches seront installés. Cela aura pour effet direct de diminuer le coût total de déploiement. Au contraire, l'augmentation de la capacité de streaming (et donc, de traitement) permet une meilleure rentabilité de ceux-ci sans devoir en ajouter et a également pour effet de diminuer le coût global de déploiement.

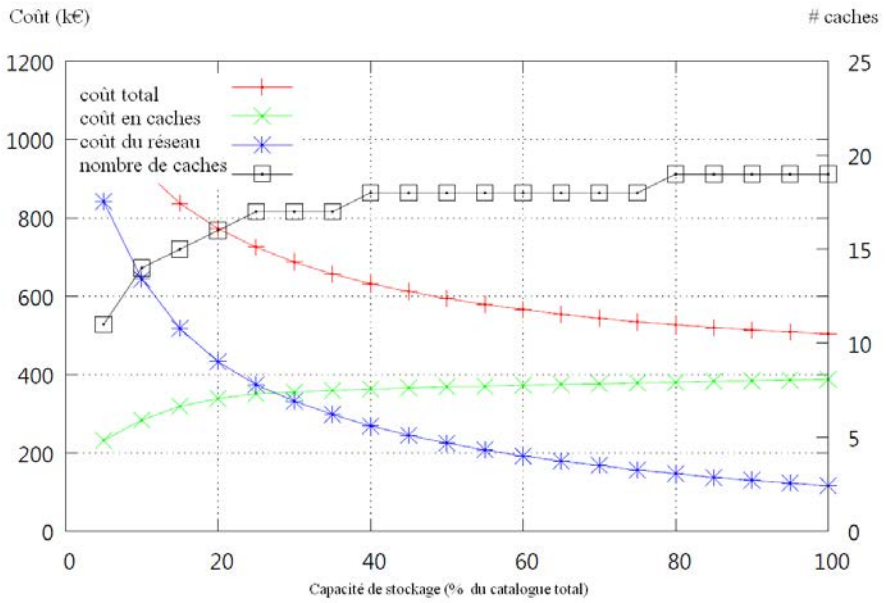



FIGURE 3.16 – *Coûts des solutions optimales (axe à gauche) et nombres de caches installés dans celles-ci (axe à droite), lorsque la capacité de stockage unitaire de cache varie de 0 à 100 % (valeur de référence à 20 %).*

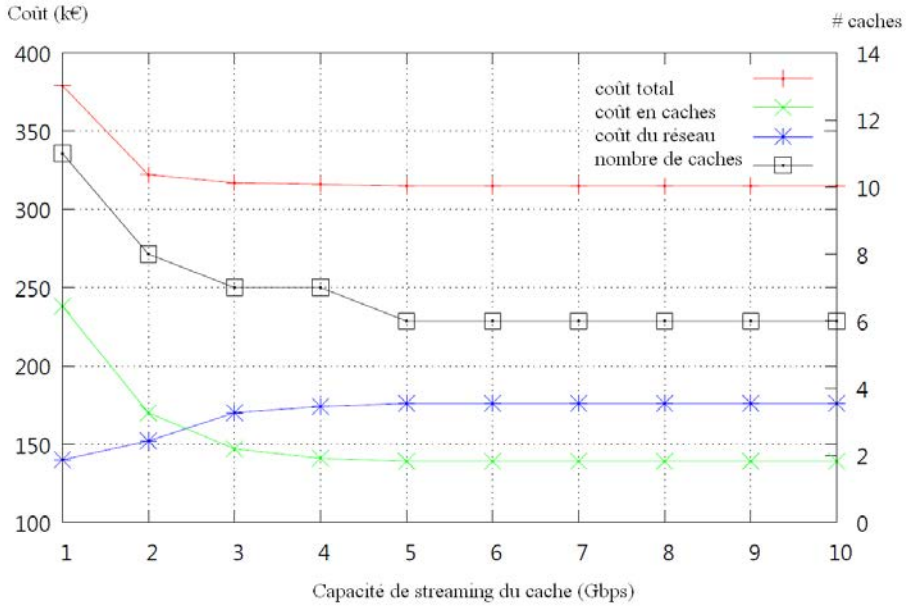



FIGURE 3.17 – *Coûts des solutions optimales (axe à gauche) et nombres de caches installées dans celles-ci (axe à droite), lorsque la capacité de streaming unitaire de cache varie de 1 à 10 Gbps (valeur de référence à 2 Gbps).*

La figure 3.18 montre le gain obtenu en optimisant le déploiement, par rapport à une architecture pré-déterminée "intuitive" (au regard des caractéristiques des caches et du trafic). Les instances testées sont ici symétriques au sens du nombre de fils (premier nombre de l'index des abscisses) et de la profondeur (second nombre de l'index des abscisses). Les architectures prédéterminées consistent ici à placer un cache à chaque nœud feuille (niveau maximal $d$), un cache à chaque nœud précédent une feuille (niveau $d-1$) ou les deux. Les écarts peuvent alors varier de 10 à 100%.

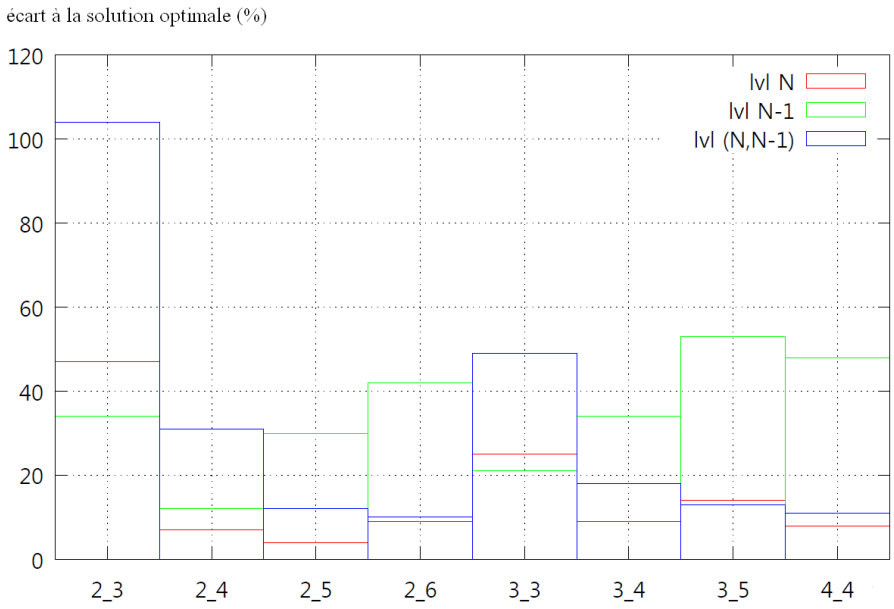



FIGURE 3.18 – *Ecart (en %) de la solution optimale avec la solution où tous les caches sont déployés aux niveaux $d$, $d-1$ ou les deux.*

### 3.8.2 Configurations de cache et scénarios de coûts différents

Dans la pratique, les fournisseurs de caches peuvent proposer des modèles aux caractéristiques différentes mais les choix restent souvent relativement limités. Pour le déploiement de caches pour la vidéo à la demande nous allons nous concentrer sur l'impact de ces caractéristiques sur le déploiement optimal, et les recommandations économiques suivant des scénarios envisagés.

Nous supposerons ici que le fournisseur de contenus peut choisir jusqu'à deux types d'équipements parmi une gamme restreinte, et a la possibilité d'établir le déploiement optimal associé à son/ses choix d'équipements. Pour pouvoir déterminer avec précision l'architecture optimale associée à ces choix, nous devons également estimer des coûts réalistes pour chaque pièce d'équipement. Malheureusement, malgré les sources diverses qui peuvent exister à ce sujet (www.mkomo.com/cost-per-gigabyte, www.jcmit.com/diskprice.htm,...), nous ne pouvons extraire de règles économiques claires comparant le coût de stockage et le coût de transmission en heures pleines. Pour éviter cet écueil, nous choisirons de travailler sur deux scénarios de coûts radicalement différents :

- **Scénario I** : le coût de stockage de 2 TB est équivalent au coût de transmission de 1 Gbps. En supposant un catalogue de taille (ou partie de catalogue) de 100 TB, chacune des 10 classes de contenus possède un volume de 10 TB, ce qui induit un partitionnement des 20 Keuros unitaire par cache en 3,7 Keuros (18,5 %) pour le stockage et 16,3 Keuros (81,5 %) pour la transmission.
- **Scénario II** : le coût de stockage de 1 TB est équivalent au coût de transmission de 4 Mbps. Dans ce cas, le coût unitaire de cache est partitionné entre 0,7 Keuros (3,5 %) pour le stockage et 19,3 Keuros (96,5 %) pour la transmission.

Partant de ces scénarios de coûts, nous pouvons décorréler de façon arbitraire le coût unitaire de caches et construire alors de nouveaux types de caches dédiés pour des rôles particuliers (transmission (C) ou stockage(D)), ou des caches de plus grande ou petite capacité ("Grand cache" (B) et "Petit cache" (A)), qui bénéficient de l'économie d'échelle associée. Nous considérons ici alors qu'un cache de capacité deux fois plus grande aura seulement un coût 1.5 fois plus élevé.

| **Petit cache (A)** | **Grand cache (B)** |
|---|---|
| $stockage : C_1 = 10$ TB (1 block) | $stockage : C_1 = 40$ TB (4 blocs) |
| $streaming : C_2 = 1$ Gbps | $streaming : C_2 = 4$ Gbps |
| $\forall i \in \mathcal{C}, cost(i) = 13,3$ Keuros | $\forall i \in \mathcal{C}, cost(i) = 30$ Keuros |
| **Cache dédié (C)** | **Cache dédié (D)** |
| $stockage : C_1 = 10$ TB (1 block) | $stockage : C_1 = 40$ TB (4 blocs) |
| $streaming : C_2 = 4$ Gbps | $streaming : C_2 = 1$ Gbps |
| SCENARIO I | |
| $\forall i \in \mathcal{C}, cost(i) = 16,5$ Keuros | $\forall i \in \mathcal{C}, cost(i) = 27$ Keuros |
| SCENARIO II | |
| $\forall i \in \mathcal{C}, cost(i) = 29,5$ Keuros | $\forall i \in \mathcal{C}, cost(i) = 14$ Keuros |

Le premier type d'expérimentations s'articule autour des architectures utilisant les caches R, A et/ou B. La figure 3.19 résument ces résultats. Nous pouvons constater que les caches de petites tailles (A) sont peu utiles, malgré leurs souplesse. Les meilleures solutions consistent à utiliser une combinaison de deux types de caches. Même si les valeurs sont très proches, la meilleure combinaison semble se situer avec une architecture alliant les caches de grande taille mais de coût plus rentable (B), avec le cache de référence (R).

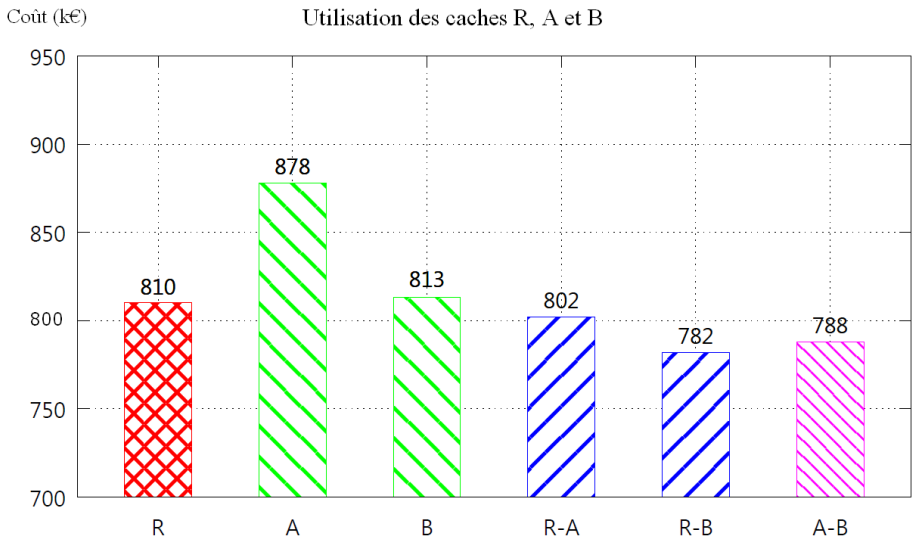



FIGURE 3.19 – *Coûts des solutions optimales utilisant un ou deux types de caches parmi R, A et B.*

La figure 3.20 montre le déploiement optimal d'architectures utilisant les caches R, C et/ou D. Le scénario I (où le stockage mémoire est le plus onéreux) rend le cache (C) utile, alors que le (D) ne semble pas être un choix judicieux. Globalement, le seul cache (R) semble suffire pour produire ici une bonne solution. Le couple de caches (C) et (D), semble un choix relativement judicieux, mais reste 10% plus cher que l'architecture précédente.

Au contraire, le scénario II (où le stockage de transmission est le plus onéreux) offre un intérêt majeur au cache (D) qui devient un choix préférable à (R), et permet à lui seul d'être un choix quasi optimal pour le déploiement de l'architecture.

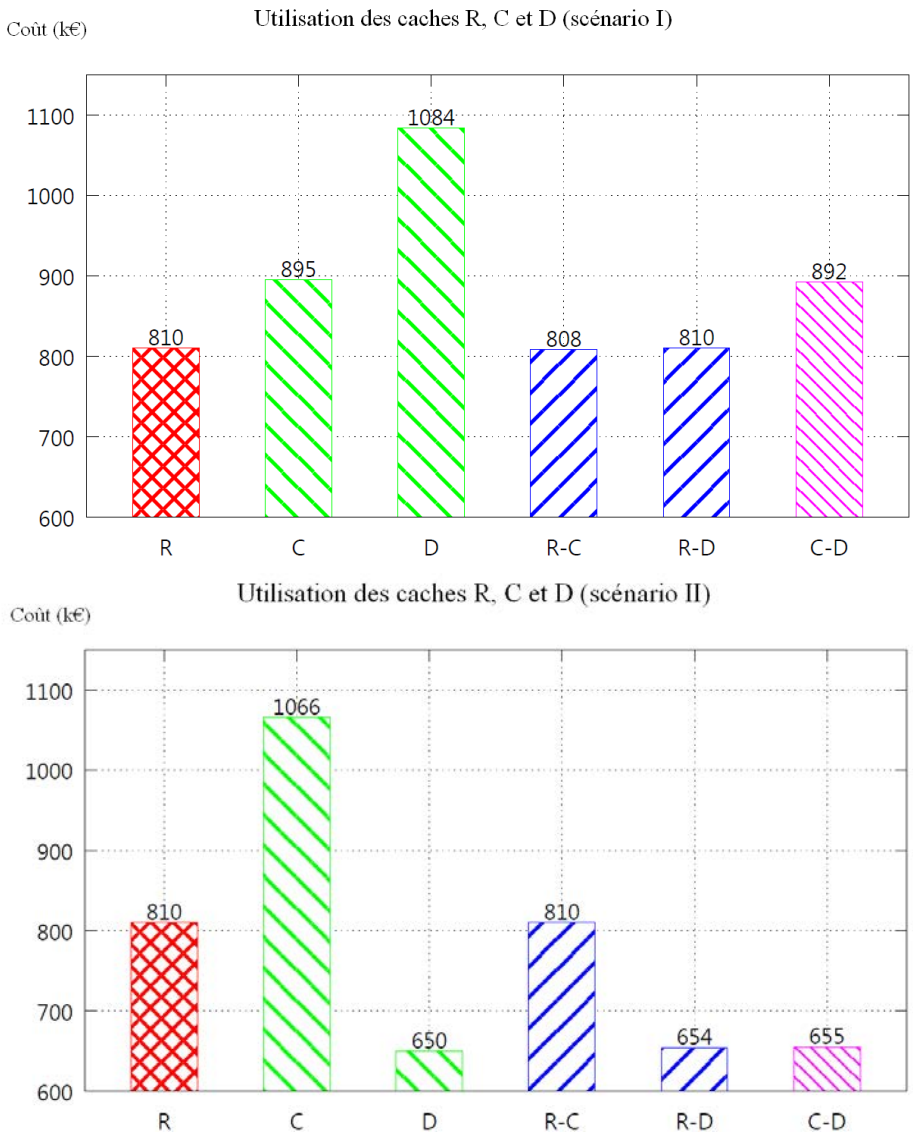



FIGURE 3.20 – *Coûts des solutions optimales utilisant un ou deux types de caches parmi R, C et D dans les deux scénarios.*

Ceci est confirmé par le nombre de caches déployés dans l'architecture optimale de la figure 3.21. Dans le scénario I, les contributions du cache (C) dans le couple (R)-(C) sont minimes, mais deviennent prépondérantes dans le couple (C)-(D). Dans le scénario II, le cache (D) est extrêmement intéressant ; au point que lorsque ceux-ci peuvent être choisis, ils sont installés abondamment.

De manière plus générale l'hypothèse du scénario II est extrêmement profitable grâce au cache de type (D), qui surclasse largement le cache de référence (R) en intérêt/efficacité. Celui-ci possède un coût principalement généré par son unité centrale (CPU), qui semble suffisamment équipé

pour délivrer efficacement les 40 TB de données qu'il contient. Dans ce scénario sa configuration semble adaptée au contexte, à la fois en coût et en souplesse. Dans l'autre scénario (scénario I), c'est le cache de référence qui semble le plus à propos pour constituer le plus gros du déploiement.

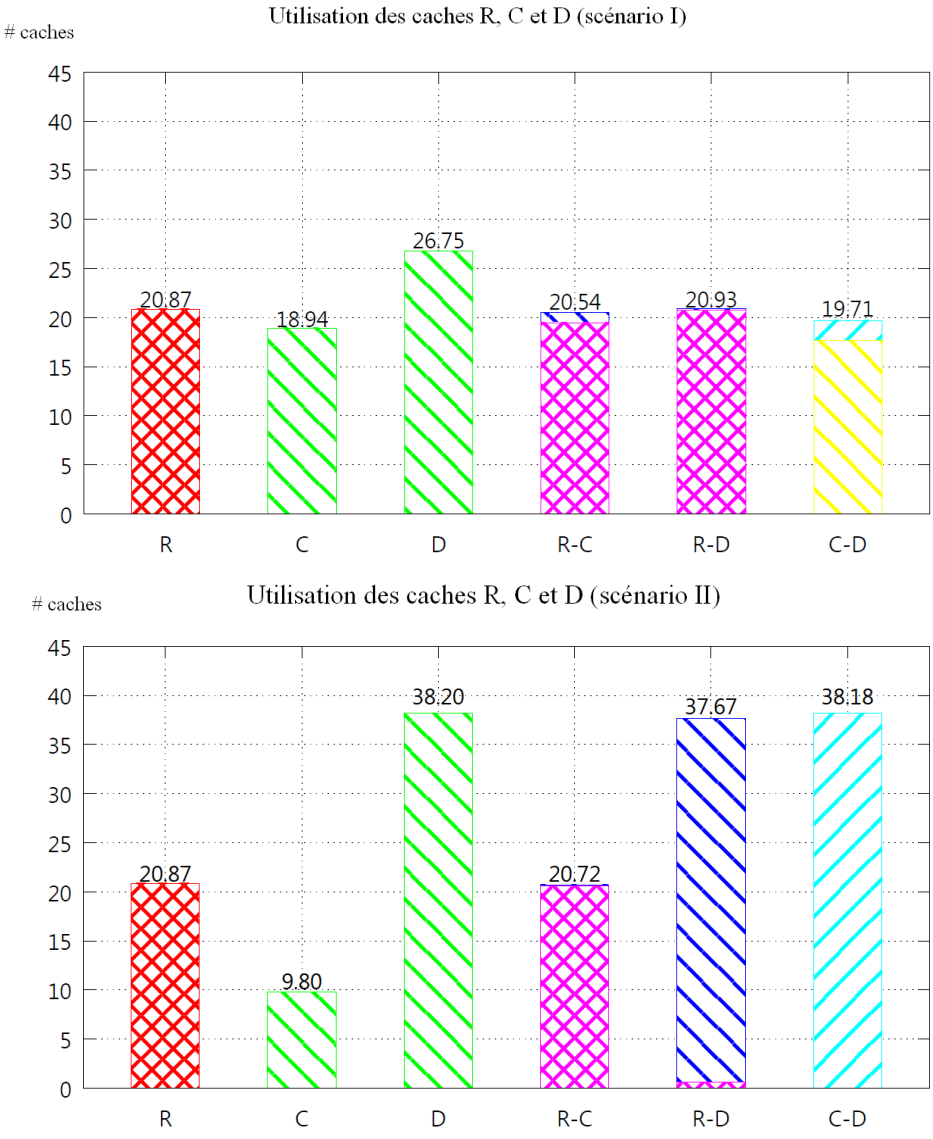



FIGURE 3.21 – *Nombre de caches déployés dans les solutions optimales utilisant un ou deux types de caches parmi R, C et D dans les deux scénarios (avec le détail pour chaque type).*

Nous avons trouvé de bonnes configurations parmi celles proposées, mais nous aimerions connaître quel serait le déploiement optimal de chacun des dispositifs suivant des hypothèses différentes. Pour ce faire nous allons artificiellement découpler les dispositifs de stockage mémoire et de transmission ; il est alors possible de composer la configuration de son choix à chaque nœud

du graphe. Nous allons alors faire varier l'impact du coût du dispositif par rapport à l'autre (consignés dans la figure 3.22 en %).

Les courbes nous révèlent d'abord les facteurs limitant de chaque configuration : à gauche de la courbe les dispositifs de stockage sont "gratuits", seuls compte alors la capacité de streaming. Le cache qui apporte alors le meilleur ratio coût/capacité de streaming est naturellement celui qui est préféré dans les solutions optimales (soit (C) selon le scénario I). Au contraire, à droite de la courbe, la capacité de streaming est un dispositif gratuit, donc non limitant ; seule la capacité de stockage a une importance. De nouveau, le cache qui apporte le meilleure ratio/capacité de stockage est celui qui sera logiquement préféré (à savoir (D) dans le scénario II).

Sur la courbe nous observons deux types de caches dont la variation des coûts de dispositifs internes change peu la valeur de la solution optimale de déploiement. C'est notamment le cas pour le cache (B), ainsi que le cache (D) du scénario II. Cela signifie que les configurations de caches sont équilibrées : peu importe la variation des coûts internes, le coût de la solution reste globalement le même, car les besoins (capacité de streaming/capacité stockage mémoire) sont bien adaptés à la configuration proposée.

Au contraire, le cache (D) du scénario I et le cache (C) du scénario II sont très instables.

Nous observons une diminution du coût de la solution optimale de déploiement lorsque le coût interne de la mémoire devient plus faible (proportionnellement à son coût global). Cela signifie que plus de mémoire a été installée (pour permettre plus d'interception). En effet, l'augmentation du coût du dispositif de streaming ne peut améliorer le coût de la solution optimale. Cela signifie qu'une partie de la capacité de streaming restait "inutilisée". Elle était donc surdimensionnée au préalable —ou la capacité mémoire sous-dimensionnée.

Le cas inverse n'est visible que par (D) du scénario II. Dans ce dernier cas, c'est la diminution du coût du dispositif de streaming qui améliore la solution : cela signifie que plus de capacité de streaming est installée, puisque l'augmentation du coût de la mémoire ne peut améliorer le coût de la solution optimale. Si l'augmentation de la capacité de streaming permet d'intercepter plus de requêtes ; c'est qu'il "restait" des requêtes à intercepter. On en conclut alors simplement que la capacité mémoire était surdimensionnée —ou que la capacité de streaming sous-dimensionnée.

Ces remarques sont étayées par les courbes 3.23, (avec un comportement spécifique de relaxation de contrainte aux extrêmes).
Nous venons donc à conclure que (D) du scénario II semble avoir la configuration la plus équilibrée par rapport au trafic considéré.

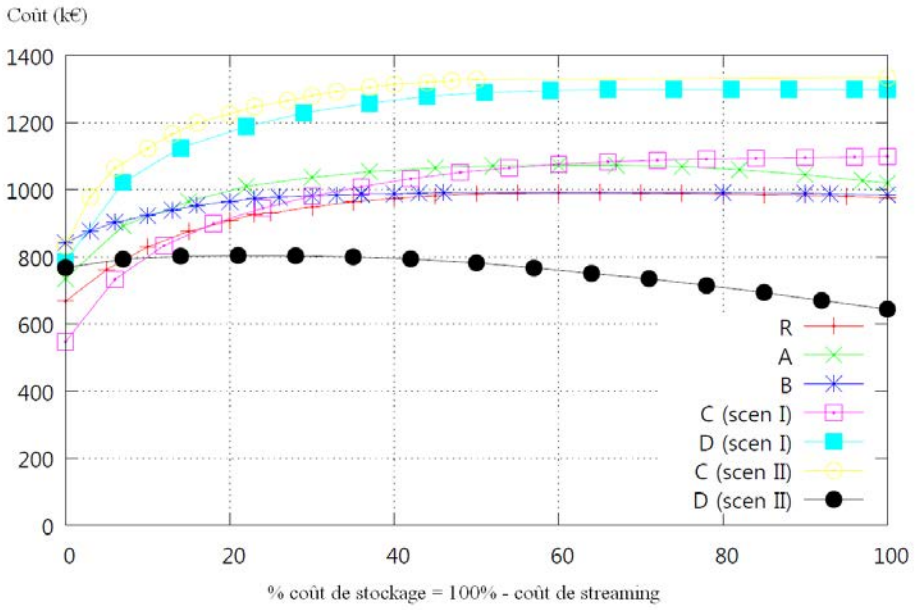



FIGURE 3.22 – *Architectures optimales lorsque les capacités de streaming et de stockage sont découplées.*

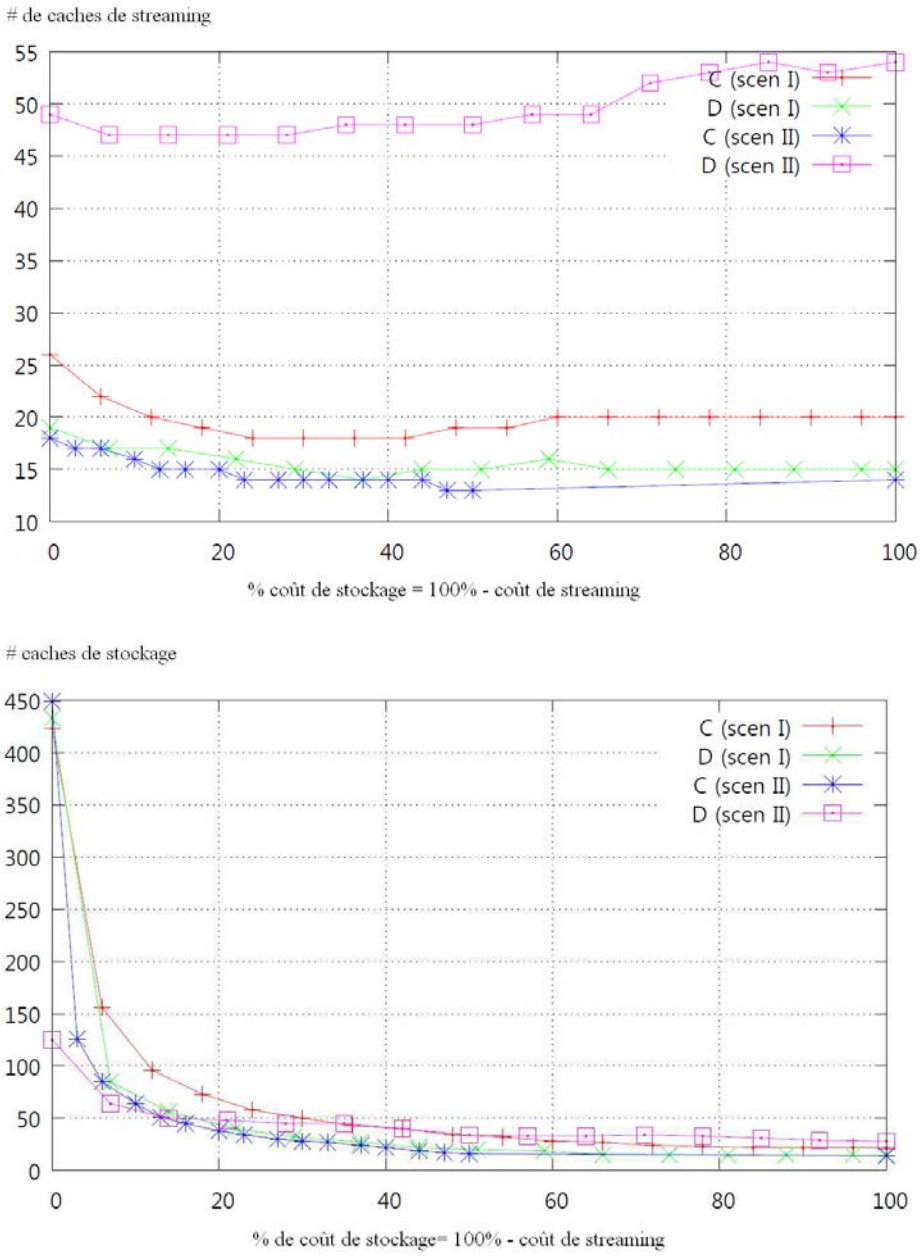



FIGURE 3.23 – *Nombre de dispositifs de stockage et de streaming installés lorsque ceux-ci sont découplés.*

## 3.9 Cas du graphe généralisé

Il est important de noter que nous n'avons pas investigué la localisation de caches à hit ratio variable dans un réseau **quelconque**, ce qui sous tendrait également un choix de routage des

requêtes. Nous pourrions alors être amené à considérer un modèle comme suit :

$$
LC \begin{cases}
\min \sum_{i \in V} C_c.y_i + \sum_{n \in N} \sum_{a \in A} w_a f_{a,n} & & (3.53) \\
\sum_{n \in N} x_{i,n} \leq C_m.y_i, & \forall i \in V & (3.54) \\
\sum_{a \in \delta^+(i)} f_{a,n} + D_i.T_f.p_{i,n} - z_{i,n} = \sum_{a \in \delta^-(i)} f_{a,n}, & \forall i \in V, \forall n \in N & (3.55) \\
\sum_{n \in N} z_{i,n} \leq y_i.C_s, & \forall i \in V & (3.56) \\
z_{i,n} \leq x_{i,n}. \sum_{j \in V} D_j.T_f.p_{i,n}, & \forall i \in V, \forall n \in N & (3.57) \\
z_{i,n} \leq x_{i,n}.D_i.T_f.p_{i,n} + \sum_{a \in \delta^+(i)} f_{a,n}, & \forall i \in V, \forall n \in N & (3.58) \\
\sum_{n \in N} f_{a,n} \leq C_a, & \forall a \in A & (3.59) \\
F_{i,n} - F_{i,n'} \geq (x_{i,n} - x_{i,n'} - 1).M_{i,n,n'}, & \forall (n,n') \in N^2, \forall i \in V & (3.60) \\
F_{i,n} = \sum_{a \in \delta^+(i)} f_{a,n} + D_i.T_f.p_{i,n}, & \forall i \in V, \forall n \in N & (3.61) \\
x_{i,n}, y_i \in \mathbf{N}, f_{i,n} \geq 0, z_{i,n} \geq 0, & i \in V, n \in N & (3.62)
\end{cases}
$$

Avec $M'$ défini comme suit :

$$M'_{i,n,n'} = \max_{n,n'} \sum_{j \in V} D_j.T_f.p_{j,n} \tag{3.63}$$

Les contraintes de conservation de flots (3.55) sont ici étendues à un graphe quelconque, les $\delta^+(i)$ et $\delta^-(i)$ représentant respectivement les arcs divergents et incidents à $i$. Le terme $z_{i,n}$ désignant toujours la variable d'interception. Les mesures de popularité, quant à elles, sont toujours représentées par les variables $F_{i,n}$, mais cette mesure est désormais effectuée sur le différentiel des demandes suivant le routage décidé.

Les solutions optimales pourraient alors, du fait de la contrainte 3.60, avoir un routage "dégénéré". Cela signifie qu'une partie des requêtes pourraient être déviée pour forcer le contenu d'un cache transparent. Pire, dans un tel modèle des circuits de flots pourraient surgir pour les mêmes raisons. La figure 3.24 représente cette pathologie.

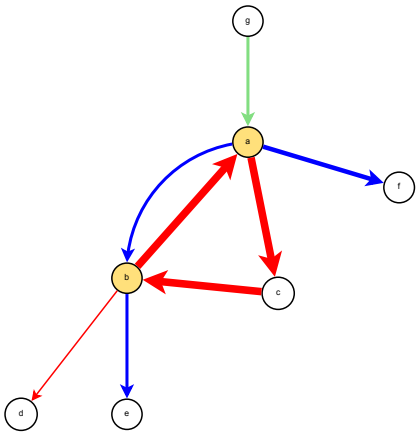



FIGURE 3.24 – *Le cache placé en b, peut voir la popularité du flot artificiellement augmenté ($F_{i,rouge}$), en appliquant un flot de même type sur le circuit $(a, c, b)$. De cette façon, s'il ne peut contenir qu'un type de fichier, et si la fonction de coût rend préférable la mise en place du contenu rouge, la contrainte (3.60) peut être ici altérée.*

On peut alors en venir à se demander si un tel problème a un sens, à la vue des solutions optimales qui pourraient survenir. De telles solutions pathologiques semblent nécessiter une information aux nœuds du réseau. On pourrait alors se dire que technologiquement, il serait dans des cas pathologiques plus judicieux de placer cette information au niveau du cache. Mais si l'on poursuit cette idée jusqu'à son terme, nous pourrons alors nous apercevoir que nous sommes en train de vouloir régir le comportement du cache en relâchant cette contrainte. Il est donc plus judicieux dans ce cas de se tourner vers une solution technologique de type serveur soumis à une intelligence globale.

## Conclusions

Le problème de localisation de caches transparents implique une modélisation du comportement des caches. Cette contrainte induit une complexité supplémentaire. Nous avons d'abord proposé un modèle de programmation linéaire en nombres entiers pour résoudre le problème. Les limites d'une telle formulation s'atteignent rapidement lorsque l'on veut résoudre des instances de taille raisonnable. Pour améliorer les performances de résolution et s'appuyer sur les spécificités de notre problème (notamment l'arborescence) nous nous sommes alors tournés vers un autre type d'approche, au lieu de considérer des renforcements de la formulation.

A partir de la littérature existante, nous avons proposé et construit une approche de résolution par la programmation dynamique. Ce paradigme permet de résoudre efficacement notre problème et reste générique pour la prise en compte de contraintes particulières. Malgré l'absence de bornes polynomiales, il reste en pratique très performant.

Nous nous sommes ensuite intéressés au déploiement optimal d'un CDN dans un contexte réaliste, et selon différentes hypothèses de scénarios (coûts, densité, caractéristiques techniques). Ceci dans le but d'exhiber les facteurs importants à prendre en considération, et les leviers d'aide à la décision.

# Conclusions et perspectives

Nous avons explicité le problème de localisation de caches sous forme de problème économique, fortement dépendant d'une estimation de coûts. Nous avons observé quelques problèmes connexes de déploiement de réseau, et notamment les problèmes de localisation dans les arborescences traités dans la littérature. Nous avons présenté une approche de programmation linéaire en nombres entiers pour résoudre le problème de localisation de caches transparents. Nous avons également exhibé une approche de programmation dynamique générique aux problèmes de localisation dans les arborescences, se basant sur un principe de récursion, non pas sur les valeurs optimales, mais sur les ensembles dominants. Cette méthode possède des performances satisfaisantes et ce, même sur des tailles d'instances élevées. Dans [73], les auteurs appliquent directement notre paradigme sur la localisation de coupleurs "PON" dans une arborescence et observent un comportement similaire. Nous pourrions étendre cette approche à des réseaux non-arborescents, en partitionnant un graphe quelconque en sous-graphes, et déterminer récursivement les ensembles dominants de solutions.

Nous avons extrait des scénarios plus ou moins réalistes afin d'avoir une vue d'ensemble des architectures de déploiement optimales, et donné les principaux leviers pour décider des investissements sur un cas d'usage réaliste. Nous remarquons de prime abord un équilibre à obtenir entre l'économie d'échelle rendant les "gros" caches plus rentables économiquement (au niveau des capacités par euro), et la souplesse offerte par les caches de taille plus modeste. Il est également préférable de bien connaître le trafic à capter afin d'équilibrer l'équipement interne du cache. Le couplage de deux équipements semble surtout efficace pour combiner la rentabilité des grands caches en zone dense, et apporter la souplesse des caches plus petits dans les zones plus disparates.

Les coûts sont extrêmement difficiles à prédire et sont fortement dépendants du contexte dans lequel nous nous situons parmi :

1. L'horizon d'investissement et/ou d'amortissement
2. L'appréciation de la qualité de service et/ou qualité d'expérience des utilisateurs, l'impact en terme de consommation
3. L'économie d'échelle
4. L'obsolescence programmée, l'avancée technologique
5. Les coûts de maintenance

Nous aimerions à présent considérer un problème connexe. Jusqu'alors, nous placions dans un horizon d'investissement relativement lointain. De ce fait, le problème visait à minimiser les coûts d'infrastructure en satisfaisant une estimation de la demande. L'enjeu était d'ordre purement financier, car toute demande formulée pouvait être satisfaite moyennant un paiement approprié.

Nous allons à présent regarder le problème sur un horizon de temps beaucoup plus court. Supposons notre infrastructure de CDN déjà établie. Nous savons que ce type de réseau possède une propension naturelle à la congestion, la latence et la perte de qualité de transmission lors des pics de trafic (période de très forte demande). Pour l'opérateur du réseau, il est important d'avoir le maximum d'informations sur les capacités de son réseau et la manière de l'exploiter le plus efficacement.

Enoncé autrement, jusqu'à quel point nous pouvons garantir la qualité de service dans le réseau actuel par une stratégie de routage optimale. Ceci nous permettrait alors de connaître les endroits où il est important d'installer des équipements (cache, changement de matériel, etc) dans le but de soulager le trafic. Cette information n'est plus liée à un problème de coût, mais de garantie de (niveau de) service.

Cette problématique est complémentaire avec les problématiques de localisation : elle permet d'aider à la décision sur la marge d'erreur et/ou l'anticipation qui est faite sur la demande. Pour illustrer son intérêt supposons une infrastructure déployée à moindre coût pour répondre à une une estimation de demande pour les trois années suivantes. Il peut être intéressant d'observer la proportion supplémentaire que nous pourrions satisfaire pour la quatrième année, ou même savoir en quelle mesure si la demande s'avérait beaucoup plus importante que prévue (risque sur l'incertitude). Si, par exemple un réseau est capable de supporter 150% de la demande maximale estimée,il existe une certaine forme de pérennité et de résilience sur ce réseau. A l'inverse, s'il s'avère que celui-ci ne peut qu'assurer une proportion maximale de 101% quelles que soient les modifications de routage que j'apporte sur mon réseau, cela signifie que mon réseau présente un risque non négligeable de saturation dans un avenir plus ou moins proche. Les investisseurs pourraient alors trouver intéressant de solliciter des équipements plus performants pour limiter ces risques.

Dans une certaine mesure, on peut voir cette réaffectation des ressources du réseau comme un recours d'une situation perturbée. Dans la littérature ce problème de routage sous contraintes de capacités est en réalité bien connu : il s'agit du "flot concurrent maximal" (FCM), que nous allons à présent étudier, ainsi que les problèmes plus généraux de multi-flots.

Deuxième partie

# Algorithmes efficaces pour le flot concurrent maximal (FCM) et décompositions

# Chapitre 4

# Les modèles de multi-flots

Dans ce chapitre, nous allons présenter une classe de problèmes classiques de la littérature, les problèmes de multi-flots. Nous nous attarderons à expliciter les modèles et les méthodes couramment utilisées pour les résoudre. Notre description s'articulera autour du problème de flot concurrent maximal, qui est le problème qui nous intéresse plus particulièrement dans toute cette partie.

## 4.1 Introduction et flot concurrent maximal (FCM)

"Comment puis-je acheminer mes biens et produits à l'aide de mes camions de sorte à ce que cela me coûte le moins cher ?" "De quelle manière dois-je construire mon réseau de fibre optique afin qu'il soit le plus efficace ?" "Quels sont les dispositifs à installer pour maintenir la pression dans mes canalisations d'eau ?"

FIGURE 4.1 – *Un réseau mondial : le réseau d'échanges d'informations.*

Ces problématiques sont rencontrées très régulièrement dans le monde industriel en général, et non seulement dans les télécommunications. Ces problématiques partagent généralement les notions de réseau, et de mouvement (de biens, d'information ou de personnes) que l'on peut modéliser par des flots. Mathématiquement, le flot représentera la quantité en circulation, tandis que le réseau sera modélisé sous forme de graphe dont les nœuds correspondront aux jonctions et les arcs aux conduits physiques. De manière plus générale, nous pouvons introduire plusieurs types de flots pour un même problème. Cette classe de problèmes est alors désignée sous la vocable de problèmes de multi-flots [3].

Si nous faisons la parallèle avec notre problématique de localisation de caches, nous voulons connaître les liens du réseau (arcs du graphes) qui sont limitants, ce qui nous donnera des informations sur les placements de caches nécessaires pour éventuellement pallier cette limitation ainsi que la criticité de notre réseau. Une autre manière de voir le même problème est de considérer la qualité de service (délai et perte) par la charge (en % de la capacité) du lien sur lequel elle transite. Un opérateur peut souhaiter connaitre la qualité de service maximale qu'il peut garantir à ses clients dans l'ensemble de son réseau. Plus exactement, connaissant le nombre de clients connectés en chaque point, quel routage de ces requêtes clients lui permettrait de minimiser la charge des liens du réseau.

Ce problème est également connu et est appelé flot concurrent maximal (FCM). Il est décrit de manière générale par un graphe, dont les arcs sont munis de capacités, et un ensemble de demandes en bien/service qui doivent être délivrées depuis leurs sources vers leurs destinations. L'objectif est alors de calculer la fraction $\gamma$ maximale (potentiellement supérieure à 1) commune à chaque demande qui peut être acheminée simultanément sur le graphe tel que le volume total transitant sur chaque lien ne dépasse sa capacité. L'exemple de la figure 4.2 montre une solution réalisable du flot concurrent maximal.

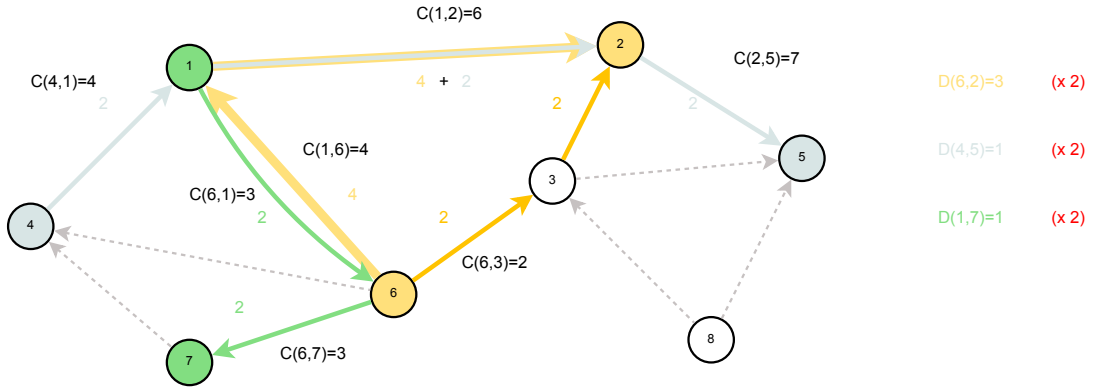



FIGURE 4.2 – *Solution réalisable du flot concurrent maximal de valeur $\gamma = 2$. Chaque fraction $\gamma$ de demande est bien acheminée sur le graphe sans violer aucune capacité (la capacité des arcs non annotés est supposée infinie).*

Le flot concurrent maximal est également le dual de la relaxation linéaire d'un problème également très connu : la coupe de densité minimale. La coupe de densité minimale est la séparation des sommets en une partition, dont le ratio de la capacité (mesurée par la somme des capacités des arcs liant les deux ensembles) sur la somme des demandes séparées par le partitionnement est minimal. L'exemple de la figure 4.3 nous montre une coupe et la valeur de densité associée. Beaucoup d'algorithmes de résolution approchée de ce problème utilisent la solution du FCM (c.f. par exemple [97]). Nous verrons plus en détail par la suite quelle importance ce problème revêt également pour notre contexte.

**Définition 27.** *Soit un graphe $G = (V, A)$ un graphe orienté muni des capacités $c_a, \forall a \in A$. Soit $T \subsetneq V$, la coupe $\Delta(T)$ est l'ensemble d'arcs défini par $\Delta(T) = \{a = uv \in A, u \in T, v \notin T\}$. Sa valeur associée est définie par $C(T) = \sum_{a \in \Delta(T)} c_a$.*

**Définition 28.** *Soit un graphe $G = (V, A)$ orienté, munis des capacités $c_{a=ij}, \forall a \in A$, et de demandes $d_{ij}, \forall (ij) \in V^2$ de source le sommet $i$ et de puits le sommet $j$. Soit $p_{i,j}$ un chemin allant de $i$ à $j$. Soit $T \subsetneq V$, une coupe de densité est une coupe $\Delta(T)$ qui a pour valeur de densité* $Sp(T) = \frac{\sum\limits_{a \in \Delta(T)} c_A}{\sum\limits_{\substack{(ij) \in V^2 \\ \nexists p_{i,j} \in G'(V, A - \Delta(T))}} d_{ij}}$.

**Définition 29.** *On appelle graphe de demandes le graphe orienté $G = (V, A)$ qui à chaque demande non nulle de sommet source $i$ et de sommet puits $j$ de $V$ associe un arc valué par cette demande $d_{ij}$.*

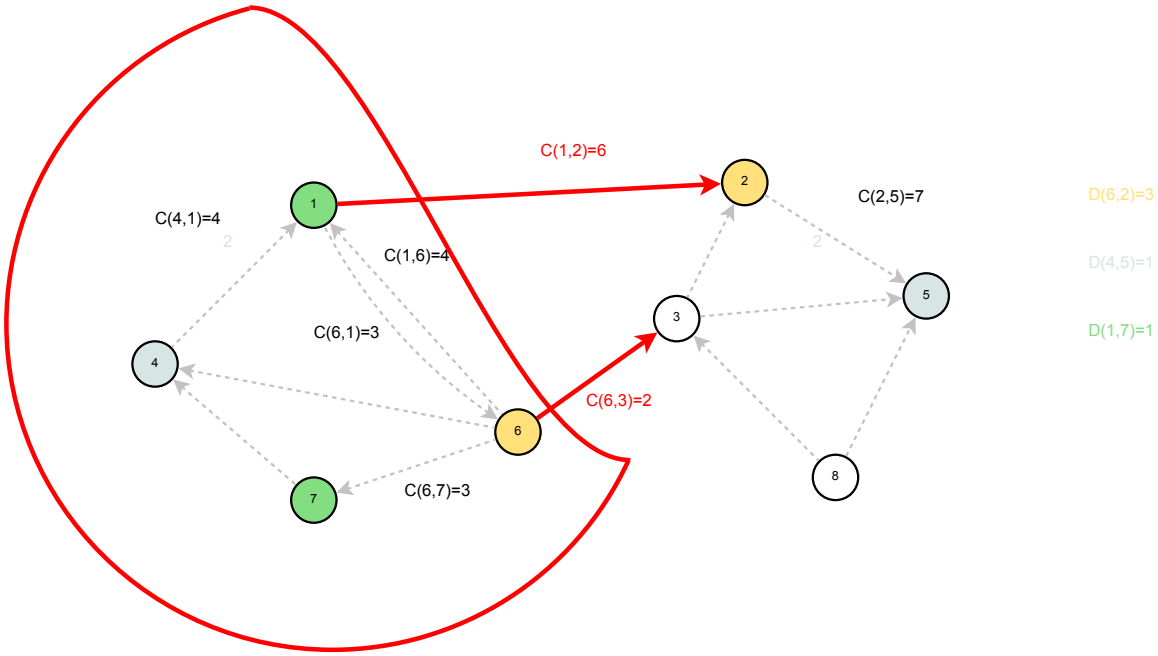



FIGURE 4.3 – *Coupe engendrée par l'ensemble $\{1, 4, 6, 7\}$. Les capacités qui séparent les partitions sont notées en rouge. Les demandes séparées par la coupe sont les demandes $d_{6,2}$ et $d_{4,5}$ ($d_{1,7}$ n'est pas séparé par la coupe) ce qui nous donne directement la valeur de la coupe de densité $\frac{8}{4} = 2$.*

## 4.2 Notations utiles

Nous décrivons ici un résumé des notations 4.1 qui seront utilisées au cours de ce chapitre. Nous allons introduire les notations suivantes. Soit $|V| = n$ et $|A| = m$, respectivement les nombres de nœuds et d'arcs. Etant donné un arc $a \in A$, nous noterons $v^-(a)$ (et respectivement $v^+(a)$) le nœud duquel l'arc diverge (et respectivement le nœud vers lequel l'arc incide). Munis d'un sous-ensemble $U \subset V$, nous noterons également $\delta^+(U)$, l'ensemble des arcs $a \in A$ divergents, donc tels que $v^-(a) \in U$ et $v^+(a) \notin U$. L'ensemble des arcs incidents sera alors caractérisé par $\delta^-(U) = \delta^+(V \setminus U)$. Lorsque l'ensemble $U = \{v\}$ n'est composé que du seul nœud $v$, nous lui préférerons la notation $\delta^+(v)$ et $\delta^-(v)$ au lieu de $\delta^+(\{v\})$ et $\delta^-(\{v\})$.

Le volume maximal de trafic qu'un arc $a \in A$ est capable de supporter ; sa capacité, est notée $c_a$ avec $c_a \geq 0$. Une demande $k \in K$ en trafic est caractérisée par une source $s^k \in V$ (l'émetteur du trafic), une destination $t^k \in V$ (le récepteur du trafic) et un volume demandé $d^k$ (parfois simplement appelée valeur de demande). Nous utiliserons également la notation $d^{ij}$, pour déterminer la valeur de demande devant être acheminée depuis $i$ vers $j$.

| Graphe orienté $G = (V, A)$ | |
|---|---|
| $V$ | Ensemble des nœuds |
| $A$ | Ensemble des arcs |
| $c_a$ | Capacité de l'arc $a$ |
| $\delta^-(U), \delta^+(U)$ | Ensemble des arcs incidents/divergents de l'ensemble de nœuds $U$ |
| $v^-(a), v^+(a)$ | Nœud depuis/vers lequel l'arc diverge/incide $a$ |
| $\delta^-(v), \delta^+(v)$ | Arcs divergents/incidents à $v$ |
| $D_G^-(v)$ | degré entrant dans $G$, cardinal de $\delta^-(v)$ dans $G$ |
| $D_G^+(v)$ | degré sortant dans $G$, cardinal de $\delta^+(v)$ dans $G$ |
| Ensemble des demandes $k \in K$, chemins et arbres | |
| $s^k, t^k$ | Source et destination d'une demande $k$ |
| $S, T(s)$ | Ensemble des sources et ensembles des destinations de la source $s \in S$ |
| $d^k, d^{ij}$ | Valeur de demande d'une demande $k$, entre $i$ et $j$ |
| $P^k$, $P^{ij}$ | Ensemble des chemins pour la demande $k$, entre $i$ et $j$ |
| $\mathcal{T}^i$ | Ensemble des arborescences de racine $i$ |
| $V_a^\tau$ | Ensemble des nœuds de $T(i)$ appartenant au sous-arbre $\tau$ de racine $v^+(a)$ |
| $lon_p^k(\mu)$ | longueur du chemin $p$ pour la demande $k$ munis des poids $\mu_a$ sur les arcs $a$. |
| $lon_\tau^{st}(\mu)$ | longueur du chemin reliant $s$ à $t$ dans l'arbre $\tau$ munis des poids $\mu_a$ sur les arcs $a$. |
| Variables de flots et variables duales | |
| $f_a^k$, $f_a$ | Flot de la demande $k$ sur l'arc $a$, le flot total sur l'arc $a$ |
| $\lambda_p^k, \beta_\tau^i$ | Proportion de flot de la demande $k$ sur le chemin $p$, sur l'arbre $\tau \in \mathcal{T}^i$ |
| $\mu_a$ | Variable duale de la contrainte de capacité en $a$ (4.24) et (5.3) |
| $\nu^k, \phi^s$ | Variable duale de la contrainte de demande en $k$ (4.25) et (5.4) |
| $\mathcal{S}$ | Solution duale, composée des vecteurs $\mu_a, \forall a \in A$ et $\nu^k, \forall k \in K$ |

TABLE 4.1 – Notations

## 4.3 Les modèles arcs-flots, appliqués au FCM

Les flots sont généralement modélisés à l'aide de variables arc-flot qui explicite la quantité de demande de chaque commodité sur chaque arc. Nous nous plaçons dans le cadre plus générique de des graphes orientés $G = (V, A)$ —où $V$ représente l'ensemble des nœuds et $A$ l'ensemble des arêtes—, plus génériques que les non-orientés (pour s'en convaincre il suffira de regarder la transformation d'une arrête en combinaisons d'arcs 4.4)

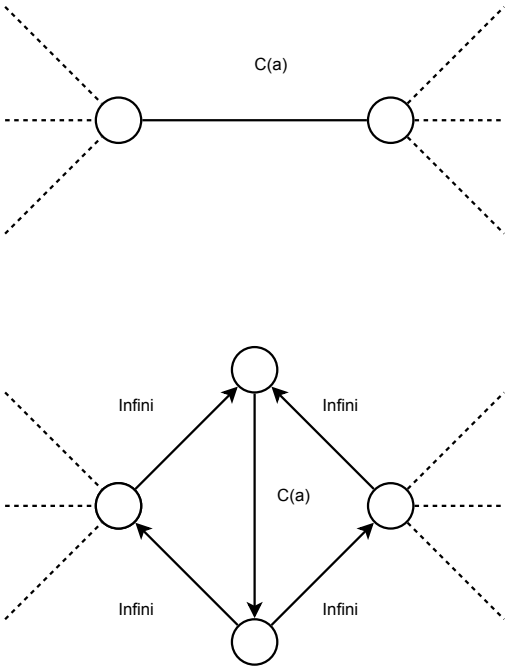



FIGURE 4.4 – *Transformation classique d'un graphe non orienté (en haut) en graphe orienté équivalent (en bas).*

Pour chaque demande $k \in K$ et pour chaque arc $a \in A$, la variable représentant la valeur de flot sera notée $f_a^k$. Nous pouvons alors modéliser le problème FCM de la manière suivante :

$$(FCM_{arc}) \begin{cases} \max \gamma_{arc}, & (4.1) \\ \sum\limits_{a \in \delta^-(v)} f_a^k - \sum\limits_{a \in \delta^+(v)} f_a^k = 0, & \forall k \in K, v \in V \setminus \{s^k, t^k\}, \quad (4.2) \\ \sum\limits_{a \in \delta^+(s^k)} f_a^k - \sum\limits_{a \in \delta^-(s^k)} f_a^k = \gamma_{arc} d^k, & \forall k \in K, \quad (4.3) \\ \sum\limits_{k \in K} f_a^k \leq c_a, & \forall a \in A, \quad (4.4) \\ f_a^k \geq 0, & \forall a \in A, k \in K. \quad (4.5) \end{cases}$$

Nous cherchons à maximiser la fraction $\gamma$ telle que $\gamma.d^k$ puisse être écoulé pour chaque demande. Les contraintes (4.2) imposent la conservation des flots : la somme des flots entrant de la demande $k$, $\sum\limits_{a \in \delta^-(v)} f_a^k$ doit égaler la somme des flots sortant $\sum\limits_{a \in \delta^+(v)} f_a^k$ de la même demande, pour chaque nœud qui n'est ni la source $s^k$, ni la destination $t^k$ de cette demande. Nous contraignons le nœud source $s^k$ de chaque demande $k$, à délivrer exactement la valeur $\gamma_{arc} d^k$ (4.8), le terminal sera implicitement contraint de recevoir la même valeur par les contraintes (4.2). Enfin, la somme des flots de toutes les demandes sur chaque arc $a \in A$, $\sum\limits_{k \in K} f_a^k$ ne doit excéder la capacité $c_a$ de cet arc, ce qui est explicité par les contraintes (4.4).

Une technique connue pour limiter le nombre de variables $f_a^k$ et de contraintes ((4.2) et (4.3)) consiste à agréger les demandes par source ou par destinations [3].

Soit $S$ l'ensemble de toutes les sources $S = \{s^k\}_{k \in K}$. Soit $T(s)$ l'ensemble des nœuds qui sont destinations et qui ont leur source en $i$ : $T(i) = \{t^k\}_{k \in K : s^k = i}$. Nous construisons alors $K^{agg}$ l'ensemble des demandes agrégées par source. Une demande $h \in K^{agg}$ est uniquement définie par sa source $s^h \in S$, et son ensemble de destinations $T(s^h)$ munies de leurs valeurs respectives de demande $d^{s^h j}$ pour chaque $j \in T(s^h)$. Le nouveau modèle aura alors la forme suivante, en nommant à présent $g_a^h$ la valeur de flots :

$$(FCM_{agg}) \begin{cases} \max \gamma_{agg}, & & (4.6) \\ \sum_{a \in \delta^-(v)} g_a^h - \sum_{a \in \delta^+(v)} g_a^h = 0, & \forall h \in K^{agg}, v \neq s^h, v \notin T(s^h), & (4.7) \\ \sum_{a \in \delta^-(v)} g_a^h - \sum_{a \in \delta^+(v)} g_a^h = \gamma_{agg} d^{s^h v}, & \forall h \in K^{agg}, v \in T(s^h), & (4.8) \\ \sum_{h \in K^{agg}} g_a^h \leq c_a, & \forall a \in A, & (4.9) \\ g_a^h \geq 0, & \forall a \in A, h \in K^{agg}. & (4.10) \end{cases}$$

Le changement principal réside dans la contrainte de satisfaction de flots qui n'est plus concentrée sur la source, mais sur chacun des terminaux (4.8), ce sont donc les contraintes d'émission de flots depuis les sources $s^h$ qui seront implicitement formulées par les contraintes (4.7).

## 4.4 Algorithmes d'approximation et méthode de déviation de flots

Le problème de FCM est un problème polynomial ; il peut donc être résolu de façon exacte en un nombre polynomial d'opérations, notamment grâce à la programmation linéaire.

**Proposition 19.** *Le problème de flot concurrent maximal est un problème polynomial.*

*Démonstration.* La formulation arc-flot (4.1) du problème continu, linéaire de flot concurrent maximal possède un nombre de variables et de contraintes polynomial. L'algorithme de l'ellipsoïde permet donc de le résoudre en temps polynomial. □

En sus, les auteurs de [19, 58] démontrent qu'il est possible de résoudre ce problème à l'aide d'un algorithme fortement polynomial. Cependant ces algorithmes ne présentent qu'un intérêt pratique limité, puisque leurs performances réelles restent très en deçà des méthodes "usuelles".

Mais même pour ce type de méthodes, les temps de résolution peuvent être très élevés, et un graphe d'une centaine de nœuds peut dans certains cas être difficile à résoudre. A titre d'exemple un graphe de seulement 125 nœuds, 600 arcs et des demandes générant un graphe complet que l'on formule par le modèle (arc-flots) induira plus de 9 millions de variables et environ 2 millions de contraintes, ce qui pourra nécessiter plusieurs dizaines d'heures pour déterminer la solution optimale.

C'est pourquoi beaucoup d'études ont été menées afin de produire des algorithmes de résolution approchée.

Shahrokhi et Matula ([76] et [77]) présentèrent l'un des premiers algorithmes d'approximation polynomiaux pour résoudre le problème de flot concurrent maximal dans un cas particulier, celui des capacités uniformes ($c_a = C_a, \forall a \in A$). Ils énoncent alors un problème qui consiste à déterminer un trafic selon des demandes, en minimisant la congestion (charge de capacité) En explicitant un modèle arc-flot pour le problème de congestion minimale (MC) nous obtenons :

$$(MC_{arc}) \begin{cases} \min \mathcal{C}_{arc}, & (4.11) \\ \sum_{a \in \delta^-(v)} g_a^h - \sum_{a \in \delta^+(v)} g_a^h = 0, & \forall h \in K^{agg}, v \neq s^h, v \notin T(s^h), \quad (4.12) \\ \sum_{a \in \delta^-(v)} g_a^h - \sum_{a \in \delta^+(v)} g_a^h = d^{s^h v}, & \forall h \in K^{agg}, v \in T(s^h), \quad (4.13) \\ \sum_{h \in K^{agg}} g_a^h \leq \mathcal{C}_{arc}.C_a, & \forall a \in A, \quad (4.14) \\ g_a^h \geq 0, & \forall a \in A, h \in K^{agg}. \quad (4.15) \end{cases}$$

L'objectif (4.11) est de minimiser la congestion maximale. Puisque les capacités sont uniformes, il s'agit alors de minimiser le flot maximum sur l'ensemble des arcs. Cette contrainte est donnée par (4.14). Les contraintes (4.12) et (4.13) assurent respectivement qu'il y a respect des conservations de flots et de satisfaction des demandes.

Ils remarquent alors que d'une solution réalisable $g^h$ de valeur $\gamma_{arc}$ non nulle du flot concurrent maximal qui possède une congestion $\mathcal{C}_{arc} = C_a$ (une capacité est au moins limitante), nous pouvons extraire une solution réalisable $g'^h_a = \frac{C_a}{\gamma_{arc}}.g_a^h, \forall a \in A$ pour $(MC_{arc})$, de valeur $\mathcal{C}_{arc} = \frac{1}{\gamma_{arc}}$. La congestion à minimiser équivaut donc à maximiser le flot concurrent maximal. De plus, on peut choisir des vecteurs solutions de flots qui caractérisent leurs solutions respectives de façon identique.

De cette proposition ils établissent une fonction de coût pour pénaliser la congestion des liens (i.e., promouvoir la répartition uniforme des flots sur le réseau). Ils l'appellent fonction de distance :

$$\Phi(a) = \frac{e^{\frac{2.|A|^2}{\epsilon}.g_a^h}}{\sum_{a' \in A} e^{\frac{2.|A|^2}{\epsilon}.g_{a'}^h}}, \quad (4.16)$$

où le paramètre $\epsilon$, est un facteur d'approximation (tolérance). Ils montrent alors que la fonction globale $\sum_{a \in A} e^{\frac{2.|A|^2}{\epsilon}.g_a^h}$ est monotone décroissante, ce qui permet de justifier la polynomialité de l'algorithme en $O(\frac{|V|.|A|^7}{\epsilon^5})$. Des travaux supplémentaires ([42], [40] et [59]) ont ensuite été effectué pour améliorer/modifier cette complexité sur ce problème et des problèmes connexes.

Cette méthode est souvent assimilée à la méthode de déviation de flot. Cette dernière méthode est directement inspirée de l'algorithme de Frank-Wolfe ([36]) et s'applique aux problèmes de routage. Proposé d'abord par ([71]), cet algorithme est surtout utile pour résoudre des problèmes

de multi-flots dont la fonction objectif est non linéaire mais différentiable dans un espace convexe fermé. La première phase, à une itération donnée, consiste alors à effectuer une approximation affine grâce au développement de Taylor au premier ordre, au point de la solution courante. Il suffit alors de résoudre le sous-problème généré par cette approximation, ce qui donne la direction pour une méthode de descente. [71] appliquent cette procédure à un problème de routage de délai minimal ($MD$) :

$$(MD_{arc}) \begin{cases} \min \vartheta = \sum_{a \in A} \frac{g_a^h}{\varpi_a - g_a^h}, & (4.17) \\ \sum_{a \in \delta^-(v)} g_a^h - \sum_{a \in \delta^+(v)} g_a^h = 0, \qquad \forall h \in K^{agg}, v \neq s^h, v \notin T(s^h), & (4.18) \\ \sum_{a \in \delta^-(v)} g_a^h - \sum_{a \in \delta^+(v)} g_a^h = d^{s^h v}, \qquad \forall h \in K^{agg}, v \in T(s^h), & (4.19) \\ g_a^h \geq 0, \qquad \forall a \in A, h \in K^{agg}. & (4.20) \end{cases}$$

La fonction objectif (4.17) modélise le délai sur chaque arc $a$ selon le paramètre $\varpi_a$, par $\frac{g_a^h}{\varpi_a - g_a^h}$, dont ils veulent minimiser la somme. Les contraintes (4.18) et (4.19) sont respectivement les contraintes de flot et de satisfaction de demandes.

De la solution courante $\tilde{g}$, ils sont capables d'extraire l'approximation affine de la fonction objectif :

$$\frac{\partial \vartheta}{\partial g_a^h}(\tilde{g}), \forall a \in A, \forall h \in K^{agg} \tag{4.21}$$

Pour améliorer la solution, les auteurs proposent alors de modifier partiellement le routage des flots de chaque agrégat séparément, afin d'améliorer la qualité de la solution (déviation de flot). L'intérêt de cette modification, est qu'elle permet de s'assurer que les contraintes (4.18) et (4.19) sont toujours respectées.

[15] s'appuie sur cette méthode et l'applique pour une résolution approchée du flot concurrent maximal. Il étend également ce principe pour la résolution approchée de programmes linéaires de façon plus générique [14]. Mentionnons aussi les travaux de [67] sur les propriétés du problème pour ces algorithmes approchés.

## 4.5 Les modèles chemins appliqués au FCM et génération de colonnes

Un résultat connu stipule également qu'un flot de $s$ à $t$ peut se décomposer sur un ensemble de chemins de $s$ à $t$, chaque chemin supportant une partie du flot. C'est une décomposition que l'on obtient notamment en effectuant la méthode de décomposition de Dantzig-Wolfe [25]. Des ensembles construits de la sorte vont implicitement satisfaire les contraintes (4.7).

L'exemple de la figure 4.5 nous montre une solution réalisable suivant le graphe donné. Chaque chemin possède sa couleur propre, la demande qui le parcourt et le facteur multiplicatif par lequel cette valeur est augmentée est de 2 ("×2"). Les demandes sont ici exprimées par le couple $(s, t)$

et la valeur de demande. La demande de 6 vers 2, est partagée entre deux chemins : le chemin "foncé" $\{6, 3, 2\}$ qui supporte une fois la demande, et le chemin "clair" $\{6, 1, 2\}$ qui supporte une fois la demande également, pour un total de $\gamma = 2$ satisfait pour cette demande, comme c'est le cas pour les autres demandes.

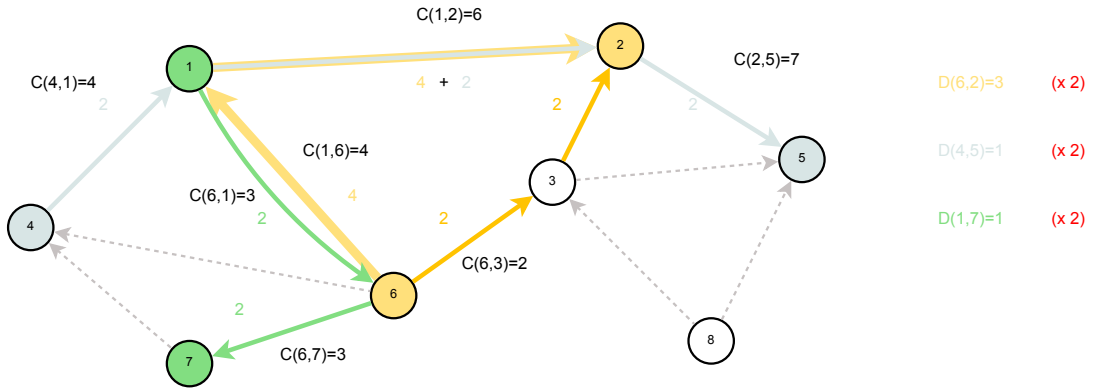



FIGURE 4.5 – *Exemple d'une solution réalisable avec chemins.*

Dans le cas du FCM, nous savons qu'il existe une solution optimale qui ne possède aucun circuit (ils peuvent être supprimés sans changer la valeur de la fonction objectif). Nous allons alors pouvoir écrire :

$$f_a^k = \sum_{p \in P^k, p \ni a} z_p^k, \ a \in A, k \in K, \tag{4.22}$$

où $P^k$ décrit l'ensemble des chemins possibles (sans cycles) reliant $s^k$ à $t^k$. $z_p^k$ représentant alors la valeur de flots routée pour la demande $k$ le long du chemin $p$. Nous pourrons dès lors acheminer les demandes $k$ selon un sous-ensemble de chemins dans $P^k$. Pour des raisons pratiques, nous utiliserons également la notation $P^{ij}$ qui décrit l'ensemble des chemins reliant $i$ à $j$.

Nous sommes alors en mesure de rappeler le modèle dit "chemin", à l'aide de variables chemins $\lambda_p^k$, représentant pour chaque demande $k$ une partie de $\gamma$ délivrée sur le chemin $p$. La somme de toutes les proportions des demandes $\lambda_p^k d^k = z_p^k$ des chemins $p$ passant par un arc $a$, ne devront alors pas violer la contrainte de capacité (4.24). Il est également important que chaque demande soit satisfaite suivant la même proportion $\gamma_{chemin}$ (contraintes 4.25) que l'on cherche à maximiser (fonction objectif (4.23)).

$$(FCM_{chemin}) \begin{cases} \max \gamma_{chemin}, & & (4.23) \\ \displaystyle\sum_{k \in K} \sum_{p \in P^k, p \ni a} \lambda_p^k d^k \leq c_a, & \forall a \in A, & (4.24) \\ \displaystyle\sum_{p \in P^k} \lambda_p^k \geq \gamma_{chemin}, & \forall k \in K, & (4.25) \\ \lambda_p^k, \gamma_{chemin} \geq 0, & \forall k \in K, p \in P^k. & (4.26) \end{cases}$$

Le problème d'une telle décomposition réside dans le nombre exponentiel de variables chemins à expliciter pour la résoudre à l'aide de la programmation linéaire. Dans ces cas-là, on utilise

généralement une procédure de génération de colonnes, héritée de la programmation linéaire généralisée, comme l'a proposée en premier [25]. Cette procédure consiste à résoudre le problème à l'aide d'un sous-ensemble restreint de variables, puis à l'aide de l'information duale, déterminer quelle(s) variable(s) pourrai(en)t améliorer le solution optimale trouvée dans cet ensemble restreint.

Dans cette formulation les variables sont associées à des chemins. Si nous écrivons le problème dual associé au FCM nous obtenons le modèle suivant. Nous appellerons $\mathcal{S}$ une solution duale qui regroupe les vecteurs $\mu_a, \forall a \in A$, et $\nu^k, \forall k \in K$.

$$\text{Dual FCM}_{chemin} \begin{cases} \min \sum_{a \in A} c_a \mu_a, & (4.27) \\ \sum_{k \in K} \nu^k \geq 1, & (4.28) \\ d^k \sum_{a \in p} \mu_a \geq \nu^k \qquad \forall k \in K, p \in P^k, & (4.29) \\ \mu_a, \nu^k \geq 0, \qquad \forall a \in A, k \in K. & (4.30) \end{cases}$$

Nous pouvons également formuler le Lagrangien, où les variables duales peuvent être vues comme des "pénalités" liées aux contraintes de capacité et de demande :

$$\min_{\mu_a, \nu^k} \max_{\lambda_p^k, \gamma_{Chemin}} \gamma_{Chemin} + \sum_{a \in A} \mu_a.(c_a - \sum_{k \in K} \sum_{p \in P^k, p \ni a} \lambda_p^k d^k) + \sum_{k \in K} \nu^k.(\sum_{p \in P^k} \lambda_p^k - \gamma_{Chemin}) \quad (4.31)$$

L'expression du coût réduit associé à une variable chemin $p$, pour une demande $k$ (ce qui est facteur de $\lambda_p^k$ dans l'équation précédente) est :

$$C(p) = \nu^k - \sum_{a \in p} \mu_a.d^k \quad (4.32)$$

Puisque le FCM est un problème de maximisation, nous devons obtenir à l'optimum, quel que soit le chemin, un coût réduit négatif ou nul. Dans le cadre d'une génération de colonnes, de façon analogue à la méthode de pivot classique du simplexe, nous cherchons généralement à trouver la/les meilleure(s) variable(s) à introduire au modèle, soit ici étant données les valeurs duales $\nu^k$ et $\mu_a$, la variable qui maximise le coût réduit (4.5), pour chaque demande $k$. Le seul critère que l'on peut faire varier ici est l'ensemble des $\mu_a$ appartenant au chemins, qui est ici précédé d'un facteur négatif. C'est donc naturellement que le problème de choix de meilleure variable à ajouter devient un problème de plus court chemin, que l'on peut résoudre par un algorithme polynomial (par exemple, Dijsktra).

L'algorithme de génération de colonnes peut être décrit par l'algorithme 11 et une illustration peut être donnée par la figure 4.6.

**Algorithm 11** Flot concurrent maximal —Génération de colonnes chemin

**Require:** $G(V, A)$ graphe orienté, $c_a, \forall a \in A$ capacités des arcs et $k \in K$

1: Calculer le plus court chemin $p^k$ pour chaque $k \in K$
2: $P^k$ ensemble courant des chemins pour chaque demande $k$
3: **repeat**
4: $P^k \leftarrow P^k + p^k, \forall k / C(p^k) > 0$ : ajouter le chemin $k$ le plus court à l'ensemble restreint des variables chemins, si son coût réduit est strictement positif
5: Résoudre le programme linéaire FCM avec $P^k$ pour chaque demande $k \in K$ de solution duale $\mathcal{S}$
6: Calculer le plus court chemin $p^k$ pour chaque $k \in K$, selon les poids $\mu_a, \forall a \in A$
7: **until** $\exists k \in K / C(p^k, \mathcal{S}) > 0$ Il existe un plus court chemin de coût réduit strictement positif, selon les variables duales $\mathcal{S}$
8: **return** $\gamma^*$ avec $P^k = P^{k*}$ pour chaque demande : les ensembles permettent de décrire une solution optimale de valeur $\gamma^*$

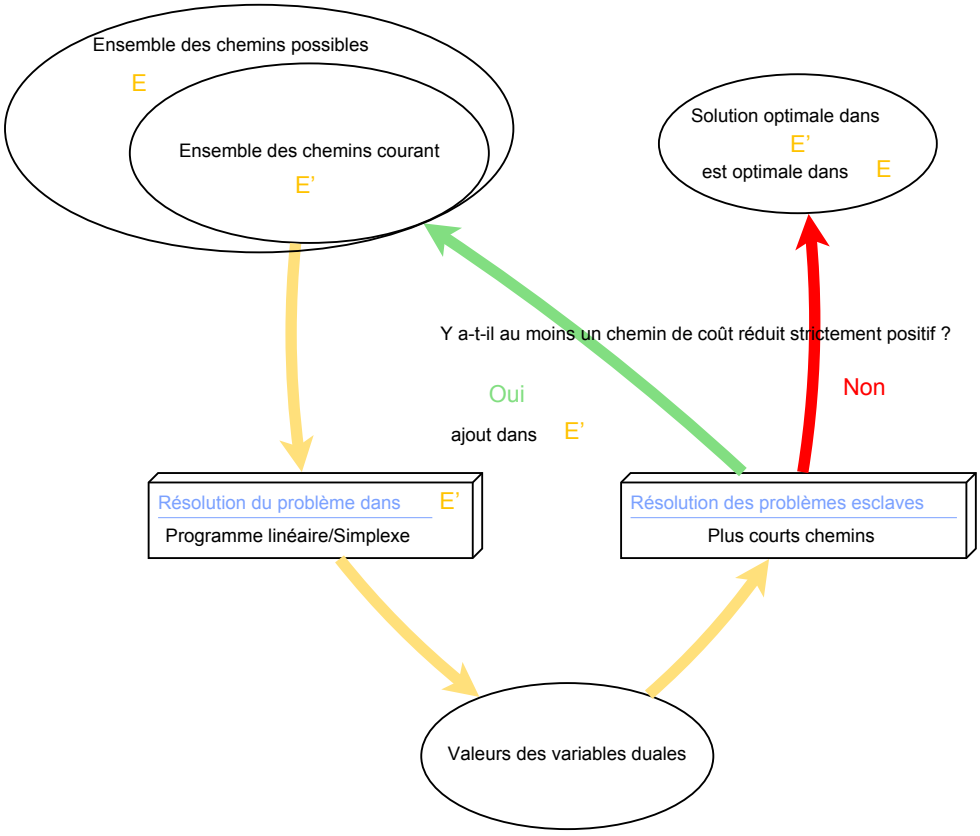



FIGURE 4.6 – *Principe de génération de colonnes, appliqué au flot concurrent maximal.*

## 4.6 Algorithme du "In-Out"

Dans [12], les auteurs proposent une amélioration de l'algorithme de génération de colonnes par hyperplans coupants valides. L'idée repose sur la distance entre deux points variables particuliers ; l'un réalisable, appelé "in" et l'autre a priori irréalisable "out". L'algorithme stoppe lorsque ces deux points sont confondus (ou que la distance les séparant est inférieure à un $\epsilon$ donné).

Cette procédure s'appliquant plus à une génération de "lignes", ou contraintes, nous pourrons utiliser la version duale de notre problème FCM, dans son modèle chemins. Les plus courts chemins au sens des valeurs duales $\mu_a$ , sont donc des coupes pour notre problème dual.

Dans un premier temps nous initialiserons une solution réalisable pour le problème dual. Une méthode pour y parvenir, est de munir la totalité des arcs à une valeur unitaire $\mu_a = 1, \forall a \in A$, et de considérer chaque plus court chemin ($p_{pcc}$) pour chaque demande. Nous sommons alors le coût de routage de la valeur de ces demandes par leur plus court chemin associé et obtenons la valeur $\sum\limits_{k \in K} d^k \sum_{a \in p^k_{pcc}} 1$. Pour satisfaire la contrainte (4.28) il suffit alors de prendre un facteur $\eta$ tel que : $\eta . \sum\limits_{k \in K} d^k \sum_{a \in p^k_{pcc}} 1 = 1$. La solution duale réalisable sera donc celle où tous les arcs sont munis de la valeur :

$$\mu_a = \eta = \frac{1}{\sum\limits_{k \in K} d^k \sum_{a \in p^k_{pcc}} 1} \tag{4.33}$$

De façon implicite la contrainte 4.29 est également respectée, car nous avons choisi les plus courts chemins, et de fait :

$$\nu^k = d^k . \sum_{a \in p^k_{pcc}} \eta \leq d^k . \sum_{a \in p^k} \eta, \forall p \in P^k, \forall k \in K \tag{4.34}$$

La figure 4.7 donne un exemple de la manière dont le calcul est fait en pratique. Les demandes cumulées en chaque arc nous donnent la valeur par laquelle multiplier la valeur duale $\mu_a$ initialisée à 1 pour tout arc $a$. Le calcul de $\eta$ sera donc constitué de 1, divisé par cette demande cumulée, soit 7. $\eta = \frac{1}{7}$. Les valeurs $\nu^k$ seront respectivement $\nu^{12} = \eta$, $\nu^{13} = \eta$, $\nu^{15} = 4.\eta$ et $\nu^{46} = \eta$. La solution est bien réalisable.

Le point "Out" est donné tout simplement par la résolution du programme linéaire (les coupes-contraintes de chemins n'étant pas toutes intégrées, nous obtenons une relaxation du problème, qui peut ne pas être réalisable).

Nous allons alors choisir un point $\mathcal{S}_{IO}$, solution constituée des vecteurs $\nu^k, \forall k \in K$ et $\mu_a, \forall a \in A$, qui est combinaisons linéaire des solutions "In" et "Out", selon un $\alpha$ choisi. En pratique nous déterminerons :

$$\nu^k_{IO} = \alpha . \nu^k_{In} + (1 - \alpha) . \nu^k_{Out}, \forall k \in K \tag{4.35}$$

et

$$\mu_{a,IO} = \alpha . \mu_{a,In} + (1 - \alpha) . \mu_{a,Out}, \forall a \in A \tag{4.36}$$

Ce nouveau point peut alors être réalisable ; dans ce cas nous pouvons mettre à jour la valeur de notre solution duale réalisable. Si ce point, au contraire, n'est pas réalisable, alors la coupe

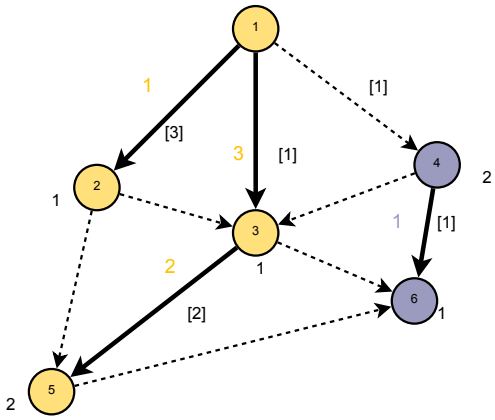



FIGURE 4.7 – *Calcul d'une solution réalisable du point de vue dual.*

du polyèdre est plus proche d'une facette du polyèdre. La figure donne l'intuition de l'efficacité d'une telle méthode 4.8.

L'algorithme adapté au FCM est présenté sur 12.

**Algorithm 12** flot concurrent maximal -Génération de colonnes chemin

**Require:** $G(V, A)$ graphe orienté, $c_a, \forall a \in A$ capacités des arcs et $k \in K$, $\alpha \in ]0, 1]$ paramètre du choix du point combiné, $\epsilon$ précision

1: Calculer le plus court chemin $p^k$ pour chaque $k \in K$
2: $P^k$ ensemble courant des chemins pour chaque demande $k$
3: Trouver une solution duale réalisable $\mathcal{S}_{In}$.
4: **repeat**
5: $P^k \leftarrow P^k + p^k, \forall k / C(p^k) > 0$ : ajouter le chemin $k$ le plus court à l'ensemble restreint des variables chemins, si son coût réduit est strictement positif
6: Résoudre le programme linéaire FCM avec $P^k$ pour chaque demande $k \in K$ : déterminer $\mathcal{S}_{out}$
7: **repeat**
8: Déterminer $\mathcal{S}_{IO} = \alpha.\mathcal{S}_{In} + (1 - \alpha).\mathcal{S}_{out}$, combinaison linéaire du point intérieur et extérieur
9: Calculer le plus court chemin $p^k$ pour chaque $k \in K$, selon les poids $\mu_{a,IO}, \forall a \in A$
10: **if** $\forall k \in K C(p^k, \mathcal{S}_{IO}) \leq 0$ **then**
11: $\mathcal{S}_{In} = \mathcal{S}_{IO}$ on met à jour le point intérieur (réalisable)
12: **end if**
13: **until** $\exists k \in K / C(p^k, \mathcal{S}_{IO}) > 0$
14: **until** $\sum_{a \in A} \mu_{a,In}.c_a - \sum_{a \in A} \mu_{a,Out}.c_a < \epsilon$ Comparaison des valeurs de solution du point intérieur et extérieur
15: **return** $\gamma^*$ avec $P^k = P^{k*}$ pour chaque demande : les ensembles permettent de décrire une solution optimale de valeur $\gamma^*$

La difficulté réside dans la façon de trouver l'$\alpha$ le plus efficace. Les auteurs de [12] donnent quelques pistes à ce sujet.

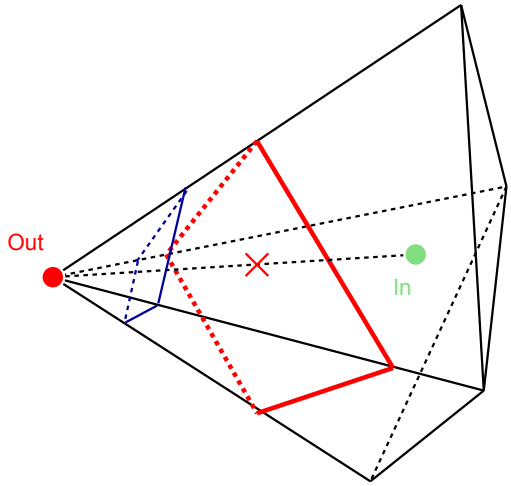



FIGURE 4.8 – *Les deux points particuliers de l'espace sont représentés en vert (Solution réalisable) et en rouge (solution optimale trouvée par l'exécution du problème relaxé). Le plan rouge représente une facette du polyèdre. Le plan bleu représente la coupe que l'on ajouterait en prenant* $\mathcal{S}_{IO} = _{Out}$. *En choisissant un* $\alpha$ *approprié nous pouvons donc potentiellement nous rapprocher de cette "frontière" de réalisabilité donnée par la facette rouge.*

## Conclusion

Les modèles de multi-flots formulent un grand nombre de problématiques industrielles, et à ce titre leurs méthodes de résolution ont beaucoup été étudiées. Nous avons notamment présenté une méthode générique pour résoudre efficacement ces problèmes de grandes tailles en introduisant successivement des variables pertinentes au modèle. Cette méthode de génération de colonnes induit implicitement une décomposition du problème. Dans beaucoup de problèmes de multi-flots la variable chemin est utilisée. Bien que son nombre peut théoriquement être exponentiel, ce type de variable contient plus d'informations (tous les arcs choisis depuis une source vers une destination), et en pratique il n'est généralement pas nécessaire d'en expliciter beaucoup pour pouvoir conclure à l'optimalité de la solution courante.

Le flot concurrent maximum est un problème facile au sens mathématique du terme, il peut donc être résolu en temps polynomial. Mais à la vue de l'importance qu'il revêt ainsi que les difficultés que l'on peut avoir à appréhender certaines instances réalistes (même des réseaux de quelques centaines de nœuds), des travaux se sont concentrés sur l'amélioration des performances de résolution par des méthodes mathématiques avancées. Nous avons également présenté quelques-uns de ces travaux.

Nous allons à présent étudier s'il existe d'autres manières de formuler un problème de multi-flots dans le but d'accélérer les processus de génération de colonnes, notamment pour celui qui nous intéresse plus particulièrement ; le flot concurrent maximum.

# Chapitre 5

# Un modèle arbre

## Introduction

Dans ce chapitre nous cherchons à améliorer l'efficacité de la génération de colonnes en utilisant des formulation implicites plus synthétiques (qui contiennent plus d'informations). Nous avons déjà vu que plusieurs demandes peuvent partager la même source ; pourtant une formulation chemin induira une génération des variables indépendamment pour chaque destination. Nous allons essayer de voir dans quelle mesure nous pouvons mutualiser ces décisions par l'élaboration d'un nouveau modèle que nous allons maintenant présenter.

## 5.1 Notations

Etant donné un nœud source $i \in S$, nous appellerons $\mathcal{T}^i$ l'ensembles des arborescences (arbres orientés) de racine $i$ et couvrant (au moins) l'ensemble $T(i)$ des destinations associées. Nous appelons $V_a^\tau$ le sous-ensemble de nœuds terminaux $j \in T(i)$, associé à l'arborescence $\tau \in \mathcal{T}^i$ et l'arc $a \in A$, tel que l'unique chemin de l'arborescence $\tau$ qui connecte $i$ à $j$ contient $a$ :

$$V_a^\tau = \{j \in T(i), \exists p \in P^{ij} \text{ tel que } p \subset \tau,\ p \ni a\}. \tag{5.1}$$

En d'autres termes, $V_a^\tau$ est l'intersection de $T(i)$ avec l'ensemble des nœuds couverts par la sous-arborescence $\tau$ de racine $v^+(a)$. Dans ce modèle, nous utilisons une variable commune $\beta_\tau^i$ pour tous les terminaux liés à une source donnée. Cette variable est alors associée à une arborescence couvrante $\tau \in \mathcal{T}^i$, et, pour chaque demande $k \in K$ qui possède sa source en $i$ ($s^k = i$), un volume de demande $\beta_\tau^i d^k$ sera acheminé à $t^k$ le long du chemin unique liant $i$ jusqu'à $t^k$ sur $\tau$.

Nous pouvons alors proposer le modèle "Arborescence" abrégée en "Arbre" :

$$
(FCM_{Arbre}) \begin{cases} \max \gamma_{Arbre}, & (5.2) \\ \sum_{i \in S} \sum_{\substack{\tau \in \mathcal{T}^i : \\ \tau \ni a}} \sum_{j \in V_a^\tau} \beta_\tau^i d^{ij} \leq c_a, & \forall a \in A, \quad (5.3) \\ \sum_{\tau \in \mathcal{T}^i} \beta_\tau^i \geq \gamma_{Arbre}, & \forall i \in S, \quad (5.4) \\ \beta_\tau^i \geq 0, & \forall i \in S, \tau \in \mathcal{T}^i. \quad (5.5) \end{cases}
$$

La somme des arborescences $\tau$ qui contiennent $a$, et la valeur cumulée des demandes liées aux terminaux de la sous-arborescence de racine $v^+(a)$, multiplié par la valeur $\beta_\tau^i$ associée représente alors la valeur effective de flot qui transite sur cet arc. Les contraintes (5.3) assurent donc que cette valeur ne dépassera pas la capacité $c_a$ de l'arc concerné. Les contraintes (5.4) nous assurent que le source va effectivement délivrer d'un facteur $\gamma$ les demandes vers toutes ses destinations : la satisfaction de chaque demande est alors implicitement contrainte par la structure même des arborescences et sa distribution des demandes sur ses arcs.

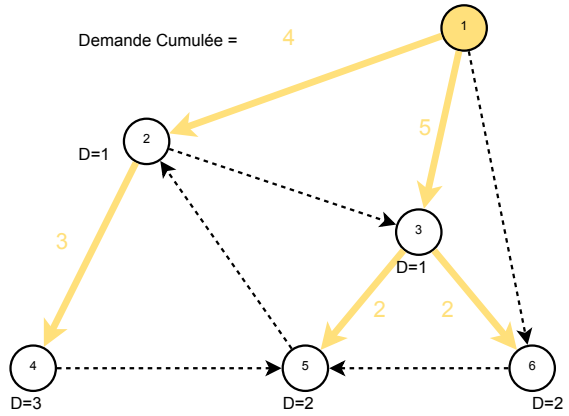



FIGURE 5.1 – *Exemple d'utilisation de variable arbre. La somme des demandes* $\sum_{j \in V_a^\tau} d^{ij}$ *qui passeront par cet arc sont notées sur l'arc en question. Ces valeurs seront multipliées par la variable* $\beta_\tau^i$ .

L'équivalence de la formulation chemin et de la formulation arbre a déjà été prouvée notamment pour le Minimum cost flow problem par [57]. La preuve s'appuie sur une décomposition de Dantzig-Wolf de la formulation arc-flots aggrégée $FCM_{agg}$. Les arbres ont également été utilisés dans le cadre d'algorithmes d'approximation comme [59]. Cependant à notre connaissance la formulation n'avait jamais été utilisée pour le FCM, et nous allons ici fournir une preuve de validité de ce modèle basée sur la dualité, et une autre preuve "constructive". Cette dernière permet notamment de construire une solution d'un modèle à partir d'une solution de l'autre L'avantage de la première preuve est qu'elle permet de prouver l'exactitude de formulations plus génériques, qui sont plus difficiles à obtenir par cette technique de décomposition. L'avantage de la seconde est qu'elle fournit un algorithme de reconstruction des arbres à partir de n'importe quelle solution du flot concurrent maximal.

## 5.2 Preuve constructive

Nous allons prouver le résultat suivant :

**Théorème 20.** $\gamma^*_{Arbre} = \gamma^*_{Chemin}$

Nous considérons à présent les terminaux $K^s$ partageant la même source ($s = s^k$ pour tout $k \in K^s$) et un flot solution $(f^k)_{k\in K}$. De plus nous supposons que les chemins $p$ supportant le flot (tels que $f^k_p > 0$) sont des chemins sans circuit. Nous appelons $G^s = (V^s, A^s)$ le sous-graphe de $G$ induit par la solution courante de flots : le nœud $v \in V \setminus \{s\}$ appartient à $V^s$ si et seulement si $\sum_{k\in K^s} \sum_{a\in\delta^-_G(v)} f^k_a > 0$ et de manière similaire un arc $a \in A$ appartient à $A^s$ si et seulement si il y a un $k \in K^s$ tel que $f^k_a > 0$. Puisque les demandes de $K^s$ partagent la même source, nous pouvons choisir un $G^s$ de valeur de solution équivalente, qui ne contient pas de circuit.

**Lemme 1.** *$G^s$ est l'union de $n^s_{\mathcal{T}}$ arborescences de racine $s$, où $n^s_{\mathcal{T}} = \prod_{v\in V^s\setminus\{s\}} D^-_{G^s}(v)$.*

Preuve : Puisque chaque nœud $v \in V^s \setminus \{s\}$ possède un degré entrant $D^-_{G^s}(v)$ positif, il suffit de garder uniquement un arc entrant en $v \in V^s \setminus \{s\}$ pour former une arborescence (de racine $s$). Il y a alors $\prod_{v\in V^s\setminus\{s\}} D^-_{G^s}(v)$ manières de construire une telle arborescence. □

Considérant à nouveau un flot $(f^k)_{k\in K}$ où toutes les demandes sont émises depuis la même source $s$, nous explicitons $f^{(s)}$ le flot résultat sur chaque arc $a$ :

$$f^{(s)}_a = \sum_{k\in K^s} f^k_a = \sum_{k\in K^s} \sum_{\substack{p\in P^k:\\ p\ni a}} f^k_p. \tag{5.6}$$

Pour chaque $v \in V^s \setminus \{s\}$ et chaque $a \in \delta^-_{G^s}(v)$, nous définissons le ratio suivant :

$$\pi^s_a = \frac{f^{(s)}_a}{\sum_{b\in\delta^-_{G^s}(v)} f^{(s)}_b} = \frac{f^{(s)}_a}{f^{(s)}(\delta^-_{G^s}(v))}. \tag{5.7}$$

**Lemme 2.** *Supposant un ensemble de demandes $K^s$, de source $s$, et une solution réalisable de flots $(f^k)_{k\in K^s}$. Alors il existe une solution de flots $(\hat{f}^k)_{k\in K^s}$ telle que :*

1. $v(\hat{f}^k) = v(f^k), \forall k \in K^s$ ;
2. $\hat{f}^{(s)}_a = f^{(s)}_a, \forall a \in A^s$ ;
3. *si* $\sum_{b\in\delta^-_{G^s}(v^+(a))} \hat{f}^k_b \neq 0$, *alors* $\pi^s_a = \frac{\hat{f}^k_a}{\sum_{b\in\delta^-_{G^s}(v^+(a))} \hat{f}^k_b}$, $\forall a \in A^s, k \in K^s$.

*où $v(f^k)$ est la valeur de flot acheminée vers $k$*

Preuve : si $G^s$ est une arborescence, alors il suffit de prendre $\hat{f} = f$. Sinon, puisque $G^s$ ne contient pas de circuit, nous pouvons construire une ordre topologique $v_1, v_2, ..., v_{|V^s|}$ où $v_1 = s$. Nous savons que $G^s$ ne contient alors pas de chemins de $v_j$ vers $v_i$ si $j > i$.

Nous construisons une solution de flots $\hat{f}$ de manière itérative. Nous prenons d'abord $\hat{f} = f$. Soit alors $v_l$ le dernier nœud (selon la relation d'ordre topologique) tel qu'il existe un arc $a \in \delta^-_{G^s}(v_l)$ et un terminal $k \in K^s$ pour lequel $\pi^s_a \neq \frac{\hat{f}^k_a}{\sum_{b\in\delta^-_{G^s}(v_l)} \hat{f}^k_b}$. Il est à présent facile de voir que tous les terminaux $k \in K^s$ qui reçoivent partiellement un flot depuis $v_l$ (i.e., ceux pour lesquels

$\sum_{b \in \delta^-_{G^s}(v_l))} \hat{f}^k_b \neq 0$) peuvent être agrégés et leur demande acheminée depuis $s$ vers $v_l$ de la même manière. Le flot $\hat{f}$ doit alors être modifié en conséquence. Nous pouvons déjà nous apercevoir que la troisième propriété est à présent satisfaite pour le nœud $v_l$. De plus, la relation d'ordre topologique nous assure que rien ne change pour les nœuds $v_j$ tels que $j > l$. Enfin, nous nous apercevons que la valeur de flot $\hat{f}^{(s)}_a$ ne change pas pour tout arc $a \in A^s$ ni pour la valeur satisfaite $v(\hat{f}^k)$ pour chaque $k \in K^s$.

Nous sommes ainsi en mesure de trouver un flot $\hat{f}$ équivalent à la solution $f$, satisfaisant les propriétés 1 et 2 de telle manière que la propriété 3 devient vraie pour un nœud supplémentaire. En réitérant cette procédure un nombre borné de fois, nous pouvons alors découvrir une solution équivalente satisfaisant toutes ces propriétés simultanément. □

Nous appellerons ici $\mathcal{T}^s(f)$ l'ensemble des $n^s_{\mathcal{T}}$ arborescences obtenues en gardant un seul et unique arc incident à $v \in V^s$. Il est important de remarquer que $\mathcal{T}^s(f)$ est le sous-ensemble $\mathcal{T}^s$ (l'ensemble des arborescences de racine $s$) induit par la solution $(f^k)_{k \in K^s}$.
Nous munissons chaque arborescence $\tau \in \mathcal{T}^s(f)$ du poids :

$$\lambda^s_\tau = \prod_{a \in \tau} \pi^s_a. \tag{5.8}$$

Le principal résultat qui nous permettra de conclure le théorème 20 est le suivant :

**Théorème 21.** *(Théorème de décomposition) Si $\varphi^{(s)}_\tau$ représente le flot sur un arbre $\tau \in \mathcal{T}^s(f)$ en acheminant $\lambda^s_\tau d^k$ pour chaque terminal $k$, entre $s$ et $t^k$, alors*

$$\sum_{\tau \in \mathcal{T}^s(f)} \varphi^{(s)}_\tau = f^{(s)}. \tag{5.9}$$

Preuve : D'après le lemme précédent, nous pouvons transformer une solution de flots $f$ en une solution $\hat{f}$ satisfaisant les trois propriétés. Puisque la transformation ne change pas le flot total $f^{(s)}_a$ sur chaque arc $a$, les proportions $\pi^s_a$ reste non modifiées également. Nous pouvons alors immédiatement dire que $f$ satisfait également la troisième propriété (i.e., toutes les demandes acheminées par le nœud $v$ peuvent être supposées acheminées de la même manière de $s$ vers $v$).

Nous allons établir la preuve du théorème par induction. Remarquons que ce résultat est vrai si $\mathcal{T}^s(f)$ est déjà une arborescence (dans ce cas $\pi^s_a = 1$ et $\varphi^{(s)}_\tau = f^{(s)}$). Considérons à présent $G^s$, son flot $f$ et ses demandes $K$ regroupés dans le triplet $\mathcal{I} = (G^s, f^{(s)}, K)$. Puisque le graphe $G^s$ n'est pas une arborescence, nous pouvons choisir un nœud particulier $v^\circ \in V^s$ dont le degré entrant $d^-_{G^s}(v^\circ) > 1$.

Décomposons à présent $\mathcal{I}$ en $d^-_{G^s}(v^\circ)$ triplets $(\mathcal{I}_\ell)_{\ell=1\ldots d^-(v^\circ)}$ avec $\mathcal{I}_\ell = (G^s_\ell, f^{(s)}_\ell, K_\ell)$. Le graphe $G^s_\ell$ est obtenu de $G^s$ en gardant un seul des $\ell$ arcs incidents à $v^\circ$ (nous appelons $a^\circ_\ell$ cet arc incident) : $A^s_\ell = A^s \setminus \delta^-_{G^s}(v^\circ) \cup \{a^\circ_\ell\}$. Les demandes $K_\ell$ sont les mêmes que $K$ mais le volume délivré pour celles-ci deviendra $d^k_\ell = \pi^s_{a^\circ_\ell} d^k$. Ces demandes peuvent être décomposées le long de leurs chemins dans $G^s_\ell$ :

$$d^k_\ell = \sum_{\substack{p \in P^k \\ v^\circ \notin p}} \pi^s_{a^\circ_\ell} f^k_p + \sum_{\substack{p \in P^k \\ a^\circ_\ell \in p}} f^k_p. \tag{5.10}$$

Le premier terme comprend tous les chemins qui ne passent pas par $v^\circ$ et le second contient tous les chemins contenant $v^\circ$ et passant uniquement par $a^\circ_\ell$ puisqu'il n'existe plus d'autre arc incident dans $G^s_\ell$. En utilisant la propriété que le flot doit satisfaire la troisième propriété du lemme 2, le terme $\sum_{\substack{p\in P^k\\ a^\circ_\ell\in p}} f^k_p$ est en réalité égal à $\pi^s_{a^\circ_\ell}\sum_{\substack{p\in P^k\\ v^\circ\in p}} f^k_p$. En d'autres termes, on peut écrire $d^k_\ell = \pi^s_{a^\circ_\ell}\sum_{\substack{p\in P^k\\ v^\circ\notin p}} f^k_p + \pi^s_{a^\circ_\ell}\sum_{\substack{p\in P^k\\ v^\circ\in p}} f^k_p$ ce qui nous amène à $d^k_\ell = \pi^s_{a^\circ_\ell} d^k$. Nous remarquons également que $G^s_\ell$ satisfait toujours le lemme 2. De plus, le coefficient $\pi^s_a$ pour $a \notin \delta^-_{G^s}(v^\circ)$ ne change pas : pour chaque graphe $G^s_\ell$ sur lesquels les demandes $d^k_l$ sont acheminées, le coefficient $\pi^s_a$ est exactement égal à celui qu'il était dans $G^s$. C'est encore une conséquence de la troisième propriété du lemme précédent.

Puisque $\sum_\ell \pi^s_{a^\circ_\ell} = 1$, le routage des demandes $d^k_\ell$ dans chaque graphe $G^s_\ell$ peuvent être considérées comme la conjonction de toutes les $\ell$ et donc devenir les demandes $d^k$ du graphe $G^s$. Notons que les demandes acheminées sur $G^s_\ell$ sont les demandes initiales pondérées par $\pi^s_{a^\circ_\ell}$. En répétant cette procédure pour chaque $v^\circ$ dans $V^s$ sur chaque sous-graphe généré, on atteint finalement un ensemble d'arborescences, dont chaque élément $\tau$ supporte une proportion des demandes égale à $\prod_{a\in\tau}\pi^s_a$. □

La preuve du théorème 20 découle alors naturellement en considérant une solution optimale par chemins $FCM_{Chemin}$ et sa valeur associée $(f^{*k})_{k\in K}$. Nous supposons alors que les demandes $K$ partagent la même source $s$. Nous notons qu'une solution optimale induira un graphe $G^s$ qui peut être considéré acyclique. De ce fait, le lemme 21 nous assure que nous pouvons créer $n^s_{\mathcal{T}}$ arborescences telles que, acheminer sur $\tau$ la proportion $\lambda^s_\tau d^k$ pour chaque demande $k$ de $s$ à $t^k$, va générer exactement la même configuration de flots $(f^{k*})_{k\in K}$. Il s'ensuit que la solution $FCM_{Chemin}$ est également réalisable pour $FCM_{Arbre}$. Nous avons $\gamma^*_{Arbre} \geq \gamma^*_{Chemin}$, du fait qu'un arbre peut être considéré comme une agrégation de chemins. Si les demandes $K$ ne partagent pas la même source, il suffit, dès lors, de décomposer le flot pour chaque nœud source et appliquer un raisonnement similaire.

La procédure du théorème 21 nous permet d'extraire immédiatement l'algorithme qui décompose toute configuration de flot réalisable, en un assemblage d'arborescences. Les figures 5.2, 5.3, 5.4, 5.5, et 5.6 montrent le principe de décomposition sur un exemple simple. Le nombre de ces arborescences, est potentiellement exponentiel.

Nous sommes d'abord en présence d'un graphe $G^1$ qui représente notre solution réalisable 5.2. Nous nous apercevons qu'en terme de flot, le degré entrant du nœud 6 est de valeur 2. Notre première décomposition de $G^1$ consiste en deux graphes distincts. On sépare donc en $\pi^1_{a_{16}} = \frac{1}{3}$ et $\pi^1_{a_{36}} = \frac{2}{3}$ la valeur totale de flot égale à 3. On met à jour le routage correspondant à la satisfaction des demandes (figure 5.3). $G^1_{16}$ n'est toujours pas une arborescence. Nous construisons donc une décomposition de $G^1_{16}$ en deux graphes distincts, suite au degré entrant du nœud 5 de valeur 2. On sépare donc en $\pi^1_{a_{25}} = \frac{1}{2}$ et $\pi^1_{a_{65}} = \frac{1}{2}$ la valeur totale de flot égale à 3. On met à jour le routage correspondant à la satisfaction des demandes (figure 5.4). En parallèle nous décomposons $G^1_{36}$ en deux graphes distincts, suite au degré entrant du nœud 5 de valeur 2. On sépare donc en $\pi^1_{a_{25}} = \frac{1}{2}$ et $\pi^1_{a_{65}} = \frac{1}{2}$ la valeur totale de flot égale à 3. On met à jour le routage correspondant à la satisfaction des demandes (figure 5.5). On pourra alors s'assurer que la somme de tous ces flots nous permet de retrouver la solution initiale (figures 5.6 et 5.7).

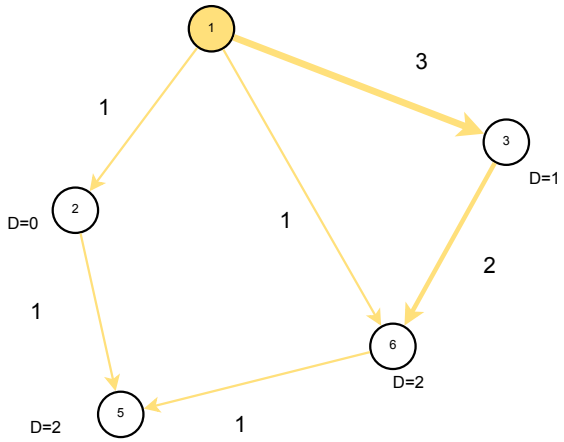



FIGURE 5.2 – *Flot initial, représenté par le graphe* $G^1$.

## 5.3 Preuve par dualité

La preuve alternative va, pour sa part, considérer les formulations primal et dual des chemins et des arborescences.

*Démonstration.* : Nous définissons $\delta_\tau^p = 1$ si et seulement si le chemin $p$ est totalement inclus dans l'arborescence $\tau$. En utilisant (5.3), nous pouvons réécrire le membre de gauche de la contrainte de capacité dans le modèle $FCM_{Arbre}$ comme suit :

$$\begin{aligned}
\text{lhs}(5.3) &= \sum_{i \in S} \sum_{\substack{\tau \in \mathcal{T}^i : \\ \tau \ni a}} \sum_{j \in V_a^\tau} \beta_\tau^i \, d^{ij}, \\
&= \sum_{i \in S} \sum_{\substack{\tau \in \mathcal{T}^i : \\ \tau \ni a}} \sum_{j \in T(i)} \sum_{\substack{p \in P^{ij} : \\ p \ni a}} \beta_\tau^i \, d^{ij} \, \delta_\tau^p, \\
&= \sum_{k \in K} \sum_{\substack{\tau \in \mathcal{T}^{s^k} : \\ \tau \ni a}} \sum_{\substack{p \in P^k : \\ p \ni a}} \beta_\tau^{s^k} \, d^k \, \delta_\tau^p, \\
&= \sum_{k \in K} \sum_{\substack{p \in P^k : \\ p \ni a}} d^k \left( \sum_{\tau \in \mathcal{T}^{s^k}} \beta_\tau^{s^k} \, \delta_\tau^p \right).
\end{aligned}$$

En posant $\lambda_p^k = \sum_{\tau \in \mathcal{T}^{s^k}} \beta_\tau^{s^k} \delta_\tau^p$, nous obtenons une solution réalisable pour le problème $FCM_{Chemin}$. En d'autres termes, étant donné une solution réalisable de $FCM_{Arbre}$, nous pouvons en extraire les $\lambda_p^k$ assurant une solution équivalente pour $FCM_{Chemin}$. Cela signifie clairement que :

$$\gamma^*_{Arbre} \leq \gamma^*_{Chemin}. \tag{5.11}$$

Pour l'inégalité inverse considérons à présent le dual de $FCM_{Arbre}$ :

$$\gamma^*_{Arbre} = \begin{cases} \min \sum_{a \in A} c_a \mu_a, & (5.12) \\ \sum_{s \in S} \phi^s \geq 1, & (5.13) \\ \sum_{t \in T(s)} d^{st} lon^{st}_{\tau}(\mu) \geq \phi^s \qquad \forall s \in S, \tau \in \mathcal{T}^s, & (5.14) \\ \mu_a, \phi^s \geq 0, \qquad \forall a \in A, s \in S, & (5.15) \end{cases}$$

avec $lon^{st}_{\tau}(\mu)$ représentant la longueur du chemin de $s$ vers $t$, dans l'arborescence $\tau$, selon les poids $\mu_a$ sur chaque arc $a \in A$. Rappelons que $(\mu_a)_{a \in A}$ et $(\phi^s)_{s \in S}$ sont les variables duales associées respectivement aux contraintes (5.3) et (5.4). De l'équation (5.14), nous pouvons voir que :

$$\phi^s \leq \min_{\tau \in \mathcal{T}^s} \sum_{t \in T(s)} d^{st} lon^{st}_{\tau}(\mu), \forall s \in S.$$

Or, puisque la somme des longueurs minimales est obtenue le long des arborescences de plus court chemin, nous avons :

$$\phi^s \leq \sum_{t \in T(s)} \min_{p \in P^{st}} d^{st} lon^{st}_{p}(\mu), \forall s \in S, \tag{5.16}$$

avec $lon^{st}_{p}(\mu)$ la longueur du chemin $p$ muni des poids $\mu_a$.
Soit $\nu^{st}(= \nu^k) = \min_{p \in P^{st}} d^{st} lon^{st}_{p}(\mu)$, nous avons alors :

$$\nu^k \leq d^{st} lon^{st}_{p}(\mu) = d^k \sum_{a \in p} \mu_a, \forall k \in K, p \in P^k. \tag{5.17}$$

De plus, grâce à (5.16) et (5.13), nous avons :

$$\sum_{k \in K} \nu^k \geq \sum_{s \in S} \phi^s \geq 1. \tag{5.18}$$

De ce fait, la solution du dual du problème $FCM_{Arbre}$ est également faisable pour le dual du problème $FCM_{Chemin}$ :

$$\gamma^*_{Chemin} = \begin{cases} \min \sum_{a \in A} c_a \mu_a, & (5.19) \\ \sum_{k \in K} \nu^k \geq 1, & (5.20) \\ d^k \sum_{a \in p} \mu_a \geq \nu^k \qquad \forall k \in K, p \in P^k, & (5.21) \\ \mu_a, \nu^k \geq 0, \qquad \forall a \in A, k \in K. & (5.22) \end{cases}$$

Nous pouvons alors écrire :

$$\sum_{a \in A} c_a \mu^*_{a,Chemin} \leq \sum_{a \in A} c_a \mu^*_{a,Arbre} \tag{5.23}$$

Nous en déduisons $\gamma^*_{Chemin} \leq \gamma^*_{Arbre}$.

□

## 5.4 Génération de colonnes appliquée à la formulation arborescente

En s'inscrivant dans une procédure de génération de colonnes comme nous l'avons déjà présenté, nous rappelons le primal et le dual du FCM dans sa version arborescente :

$$(FCM_{Arbre}) \begin{cases} \max \gamma_{Arbre}, & & (5.24) \\ \displaystyle\sum_{i \in S} \sum_{\substack{\tau \in \mathcal{T}^i : \\ \tau \ni a}} \sum_{j \in V_a^\tau} \beta^i_\tau d^{ij} \leq c_a, & \forall a \in A, & (5.25) \\ \displaystyle\sum_{\tau \in \mathcal{T}^i} \beta^i_\tau \geq \gamma_{Arbre}, & \forall i \in S, & (5.26) \\ \beta^i_\tau \geq 0, & \forall i \in S, \tau \in \mathcal{T}^i. & (5.27) \end{cases}$$

Le coût réduit associé à une variable chemin $p$, pour une demande $k$, s'écrit :

$$C(\tau) = \phi^s - \sum_{a \in \tau} \mu_a . ( \sum_{j \in V_a^\tau} d^{sj}) \tag{5.28}$$

Nous obtenons à nouveau un coût réduit dépendant uniquement de la longueur du routage des demandes ; nous maximisons donc sa valeur en prenant des plus courts chemins. Puisqu'auparavant nous devions dans le pire des cas générer l'arborescence des plus courts chemins pour chaque demande, nous remarquons ici que la complexité du problème esclave est identique. Nous pouvons donc reformuler la procédure (voir figure 5.8).

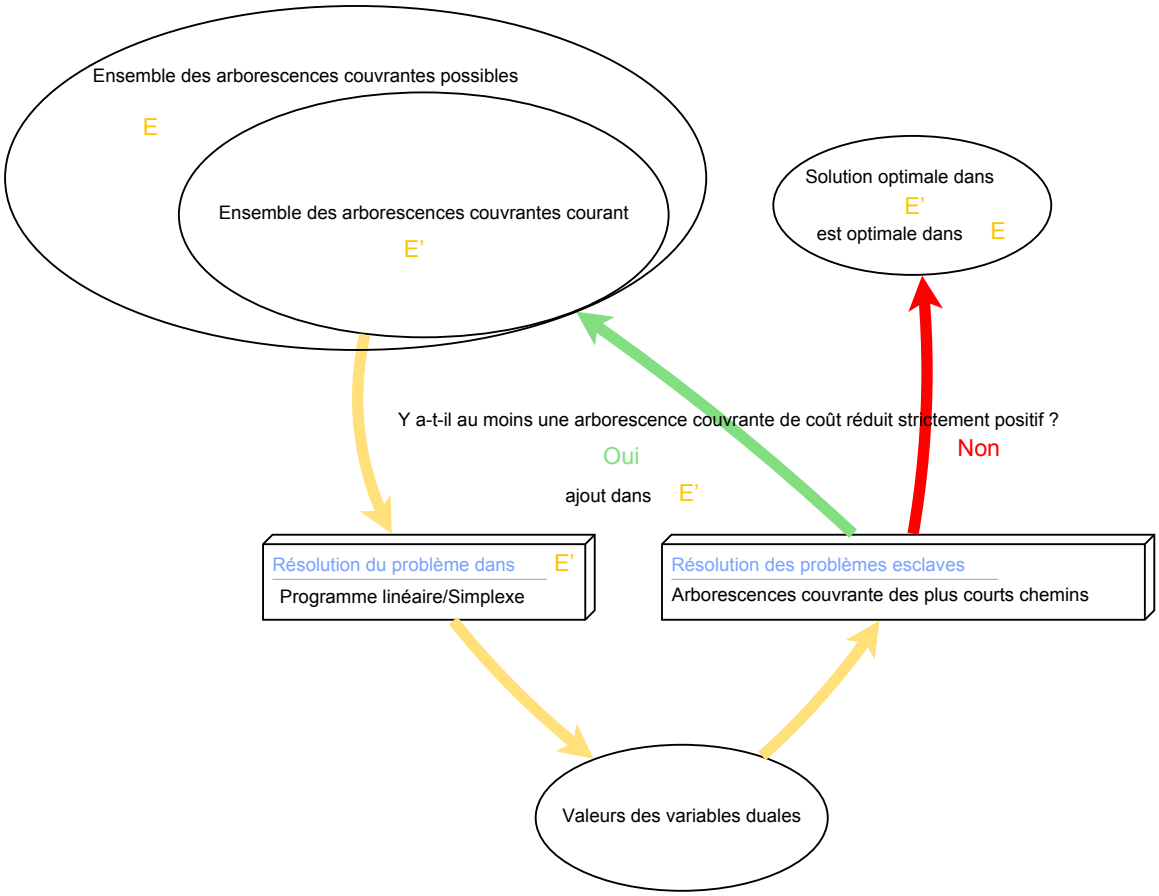



FIGURE 5.8 – *Principe de génération de colonnes, appliqué au FCM avec la formulation arborescente.*

Il intéressant de noter qu'une telle formulation présente un avantage pratique non négligeable en télécommunications ; les demandes partielles étant supportées par des arborescences, il est possible d'en extraire des tables de routage fractionnaires (puisqu'un nœud ne possède alors qu'un seul prédécesseur).

## Conclusion

Nous avons présenté ici une formulation arborescente pour modéliser les problèmes de multi-flots. Cette formulation avait déjà été extraite pour un problème de flot de coût minimal par [57]. Nous avons présenté ici une preuve d'équivalence/exactitude plus générique pour plusieurs types de formulations, ainsi qu'une preuve constructive qui explicite en détail comment créer l'ensemble d'arborescences à partir d'une solution réalisable quelconque de multi-flots. Nous l'appliquons ici à au problème de flot concurrent maximal dans le cadre d'une génération de colonnes. Cette formulation est particulièrement adaptée à un problème de télécommunications, au sens où elle forme implicitement des tables de routage "partielles".

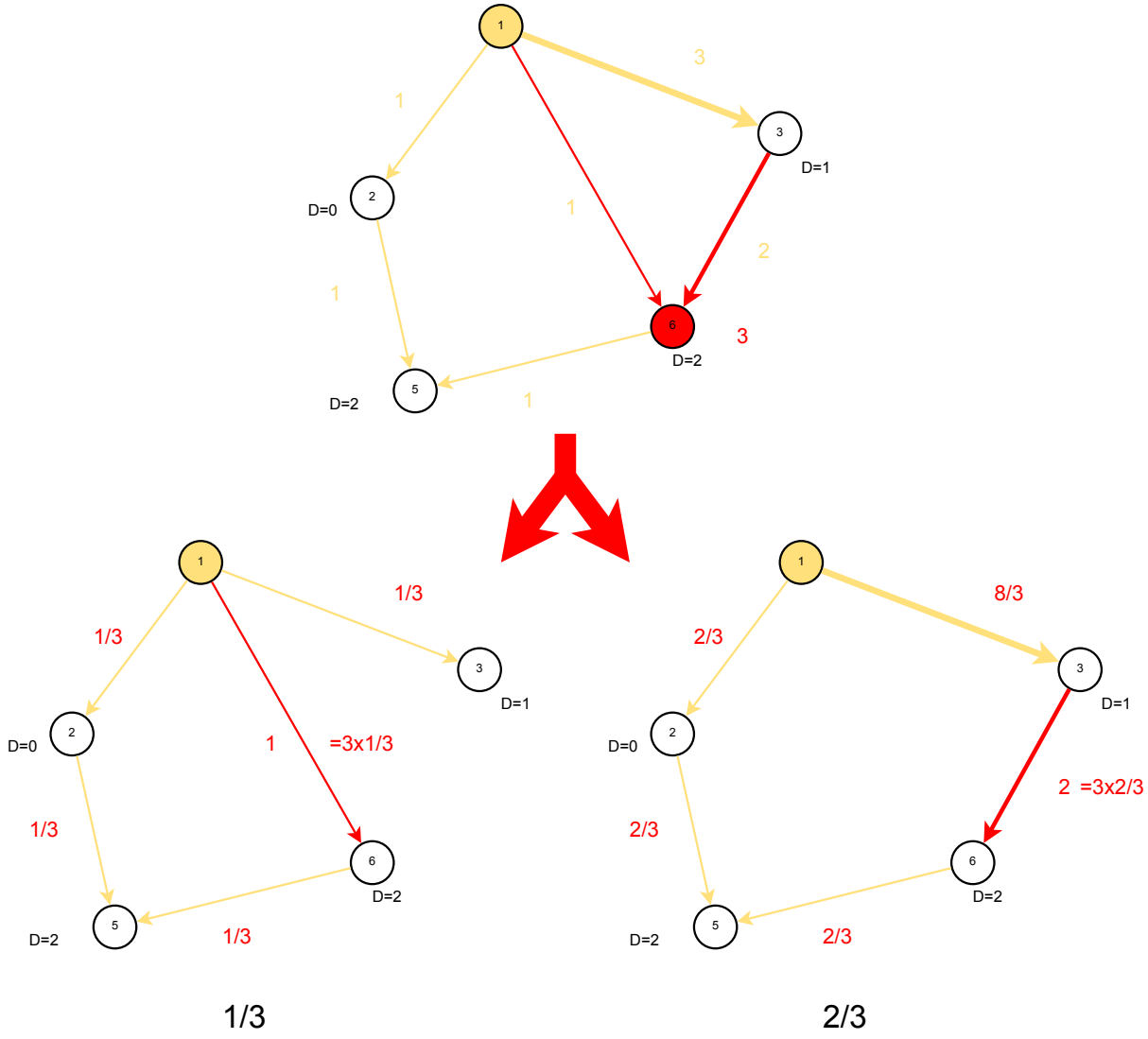



FIGURE 5.3 – *Première décomposition.*

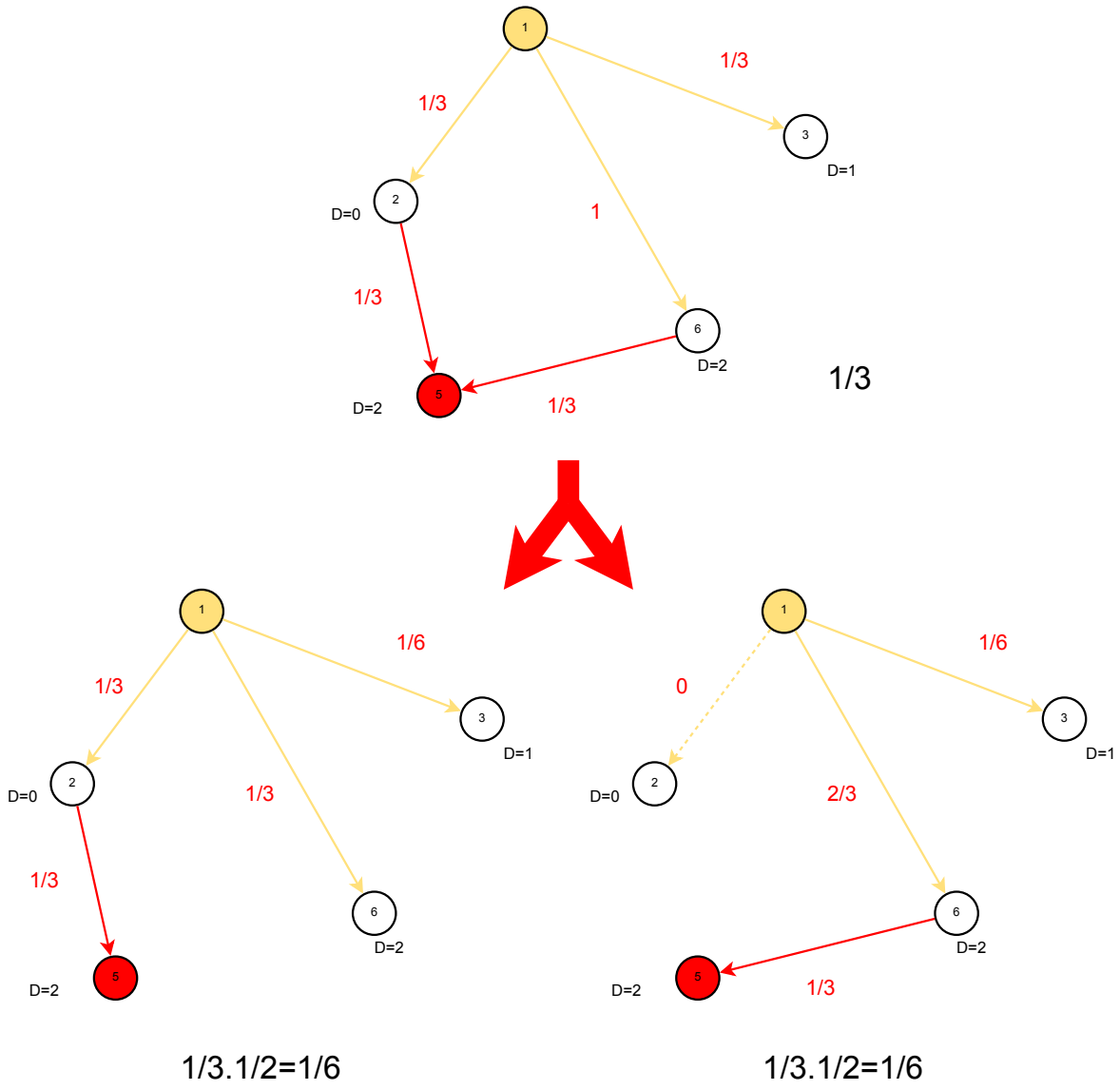



FIGURE 5.4 – *Deuxième décomposition.*

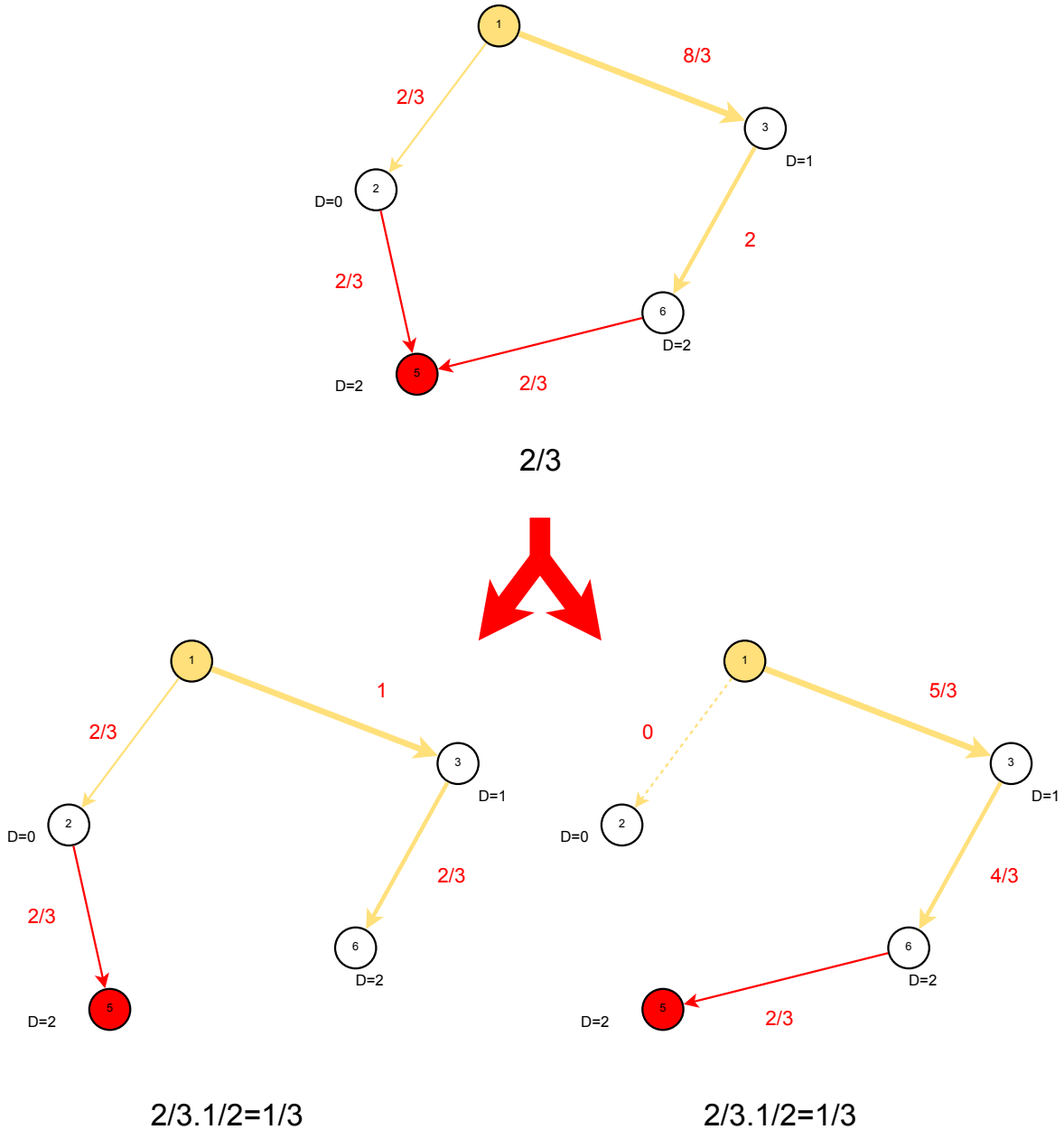



FIGURE 5.5 – *Troisième décomposition.*

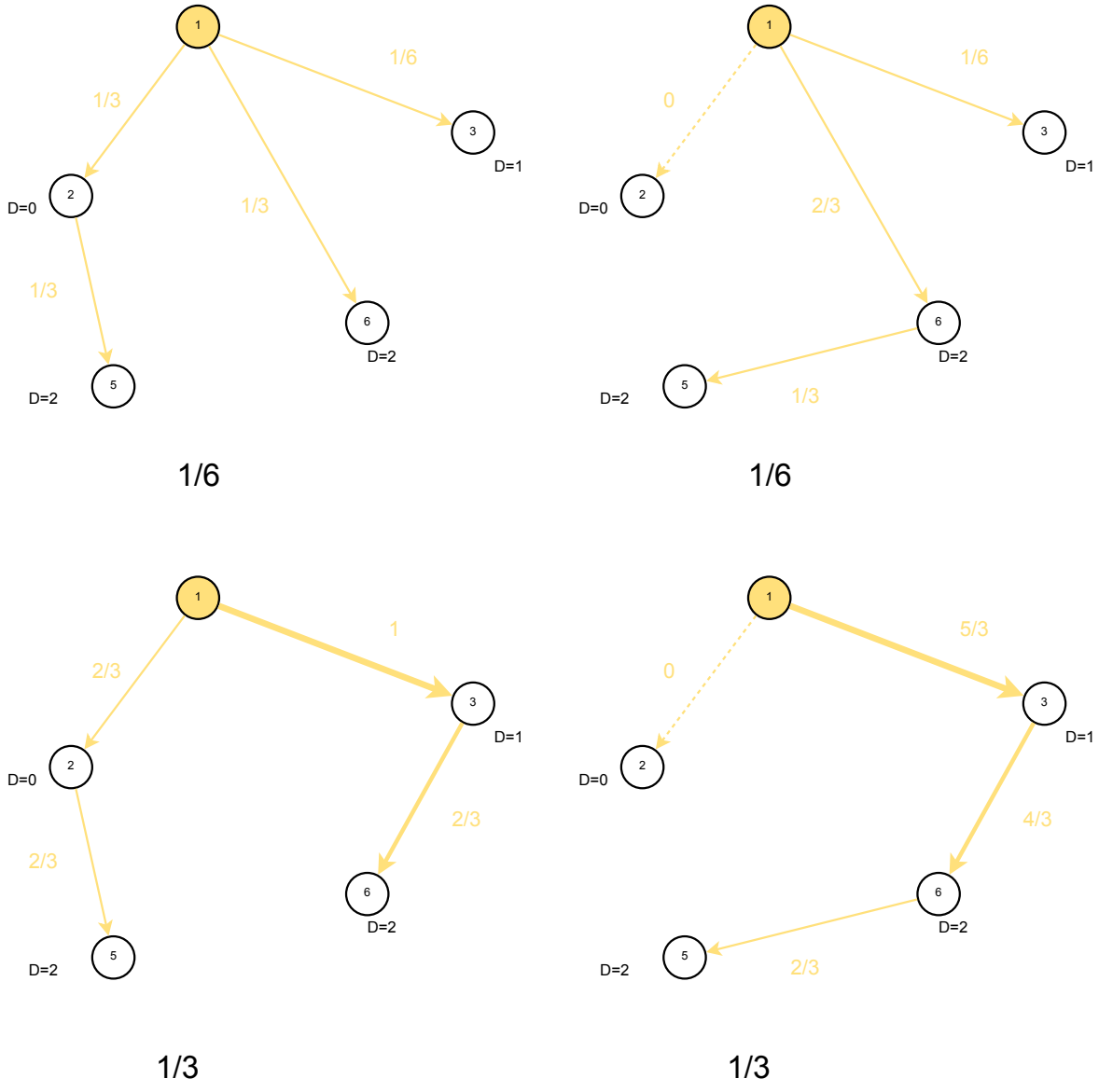



FIGURE 5.6 – *Ensemble des 4 arborescences...*

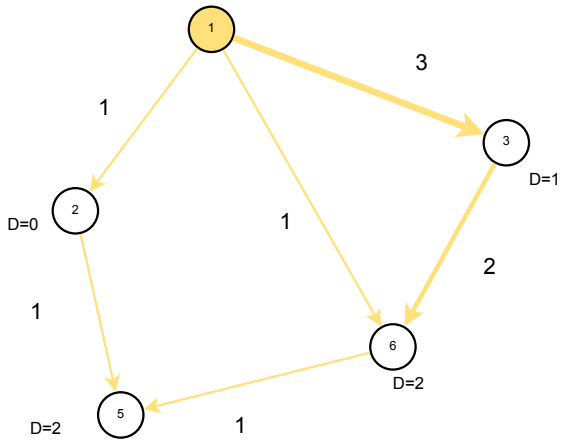



FIGURE 5.7 – *...dont l'union (la somme des flots partiels) recrée le flot initial.*

# Chapitre 6

# Formulations génériques et approches de résolution

## Introduction

Dans ce chapitre, nous allons étendre la démonstration d'exactitude à de nouveaux modèles encore plus génériques qui englobera les différentes formulations, et proposer un algorithme d'agrégation heuristique qui définit une stratégie de construction de modèle selon les paramètres de l'instance à résoudre.

## 6.1 Formulations statiques

Comme pour chaque problème de multi-flots, une solution optimale du FCM peut être décrite par un ensemble de chemins supportant les flots des sources jusqu'à leurs destinations. En suivant des méthodes de décomposition à l'instar de Dantzig-Wolfe, nous pouvons alors facilement obtenir le modèle chemins. D'autres modèles peuvent être obtenus en manipulant différemment les variables d'une formulation donnée, comme le modèle arc-flots. Une approche alternative consisterait à partir d'un modèle chemin et d'agréger des variables chemins en sous-ensembles, définissant une nouvelle variable globale. De façon générale, toute partition de l'ensemble $K$ des demandes peut induire un nouveau modèle. Considérons à présent une partition quelconque $\Gamma(K) = \{\Pi_1, \Pi_2, \ldots, \Pi_q\}$ :

$$\bigcup_{j=1}^{q} \Pi_j = K,\ \Pi_j \cap \Pi_\ell = \emptyset, \forall (j,\ell) \in \{1, ..., q\}^2 j \neq \ell. \tag{6.1}$$

Pour chaque sous-ensemble $\Pi_j \in \Gamma(K)$, nous définissons un ensemble associé $\Upsilon_j = \prod_{k \in \Pi_j} P^k$. Chaque élément de $\Upsilon_j$ est un vecteur de chemins, avec un chemin pour chaque demande $\Pi_j$. Dans notre nouveau modèle, nous associons une variable différente $\alpha_j^\varrho$ pour chaque vecteur $\varrho$ de chemins

de chaque $\Upsilon_j$. Un modèle générique peut alors être construit pour le FCM :

$$(FCM_{Gener}) \begin{cases} \max \gamma_{Gener}, & (6.2) \\ \sum_{j=1}^{q} \sum_{\varrho \in \Upsilon_j} \alpha_j^{\varrho} \sum_{k \in \Pi_j} \sum_{\substack{p \in \varrho \cap P^k \\ p \ni a}} d^k \leq c_a, \qquad \forall a \in A, & (6.3) \\ \sum_{\varrho \in \Upsilon_j} \alpha_j^{\varrho} \geq \gamma_{Gener}, \qquad \forall j = 1, \ldots, q, & (6.4) \\ \alpha_j^{\varrho} \geq 0, \qquad \forall j = 1, \ldots, q, \varrho \in \Upsilon_j. & (6.5) \end{cases}$$

Cette formulation générique contient intrinsèquement une très grande variété de modèles, incluant le modèle chemin classique déjà présenté (en prenant simplement $\Gamma(K) = K$) et le modèle arbre (en prenant une partition $\Gamma(K)$ où chaque sous-ensemble contient toutes les demandes partageant la même source).

La version duale du modèle général nous est alors donnée par :

$$\gamma^*_{Gener} = \begin{cases} \min \sum_{a \in A} c_a \mu_a, & (6.6) \\ \sum_{j=1}^{q} \psi^j \geq 1, & (6.7) \\ \sum_{k \in \Pi_j} d^k lon^k_{\varrho}(\mu) \geq \psi^j \qquad \forall j = 1, \ldots, q, \varrho \in \Upsilon_j, & (6.8) \\ \mu_a, \psi^j \geq 0, \qquad \forall a \in A, j = 1, \ldots, q, & (6.9) \end{cases}$$

où $lon^k_{\varrho}(\mu)$ représente la longueur du chemin pour la demande $k$, sur la structure $\varrho$, avec les poids $\mu_a$ sur les arcs. Rappelons que $(\mu_a)_{a \in A}$ and $(\psi^j)_{j=1,\ldots,q}$ sont les ensembles des variables duales associées respectivement aux contraintes (6.3) et (6.4).

De manière analogue à la formulation arborescente, nous pouvons montrer que le modèle général est valide. La preuve de dualité du théorème 22 ci-dessous est assez proche de celle du théorème 20 (elle sera présentée ici d'une façon plus concise).

**Théorème 22.** $\gamma^*_{Gener} = \gamma^*_{Chemin}$

*Démonstration.* : Un solution optimale de la formulation $FCM_{Gener}$ peut de manière évidente être utilisée pour construire une solution de la formulation chemin $FCM_{Chemin}$ avec une valeur de fonction objectif strictement identique. C'est une relaxation triviale du problème. Cela implique de façon naturelle : $\gamma^*_{Gener} \leq \gamma^*_{Chemin}$.

Pour prouver que la relation inverse est aussi vérifiée, nous utilisons d'abord (6.8) pour déduire que :

$$\psi^j \leq \min_{\varrho \in \Upsilon_j} \sum_{k \in \Pi_j} d^k lon^k_{\varrho}(\mu), \forall j = 1, \ldots, q.$$

Puisque la somme des longueurs minimales est obtenue à l'aide de plus courts chemins, nous avons :

$$\psi^j \leq \sum_{k \in \Pi_j} \min_{p \in P^k} d^k lon^k_p(\mu), \forall j = 1, \ldots, q, \qquad (6.10)$$

Établissant $\nu^k = \min_{p \in P^k} d^k lon_p^k(\mu)$, nous avons :

$$\nu^k \leq d^{st} lon_p^{st}(\mu) = d^k \sum_{a \in p} \mu_a, \forall k \in K, p \in P^k. \tag{6.11}$$

De plus, grâce à (6.10) et (6.7), nous avons :

$$\sum_{k \in K} \nu^k \geq \sum_{j=1}^{q} \psi^j \geq 1. \tag{6.12}$$

Dès lors, la solution du dual de $FCM_{Gener}$ est également faisable pour le dual de $FCM_{Chemin}$. Il s'ensuit naturellement que $\gamma^*_{Chemin} \leq \gamma^*_{Gener}$. □

**Remarque 4.** *Le théorème 22 reste valide dans la cas d'un graphe non orienté. Nous manipulons dès lors des chaînes. La preuve du théorème 22 reste identique.*

Toute la difficulté réside dans la création d'un modèle efficace, qui sera contraint par deux objectifs opposés :

(i) Plus les modèles sont agrégés, moins il y aura de variables nécessaires pour exprimer la solution optimale, et l'on peut espérer que la taille en variables et en contraintes du problème maître demeurera faible au cours de le procédure de génération de colonnes ;

(ii) Plus il y a d'agrégations de variables, plus il devient difficile de générer les variables utiles pour décrire la solution optimale. Cela peut induire un nombre important d'itérations de la boucle de génération de colonnes (i.e. résolutions de problèmes restreints), ce qui aura à terme un effet négatif sur le temps de résolution. Ces comportements sont empiriques et il n'existe aucun moyen à notre connaissance de déterminer un schéma d'agrégation meilleur que tous les autres.

Pour le FCM, nous avons testé plusieurs modèles : le modèle chemin $FCM_{Chemin}$, le modèle arborescence $FCM_{Arbre}$, le modèle "total" $FCM_{all}$ où toutes les demandes sont agrégées en un seul ensemble, et un modèle $FCM_{\mathbf{i}tr}$ où $i$ arborescences sont agrégées (on peut se référer à la figure 6.1). Dans la section qui regroupe les expérimentations numériques, nous nous sommes particulièrement intéressé aux cas $i = 2$ ($FCM_{\mathbf{2}ar}$) et $i = 4$ ($FCM_{\mathbf{4}ar}$).

Plus généralement si l'on considère le modèle générique utilisé dans le cadre d'une génération de colonnes, et utilisant (6.8), le coût réduit général de la variable $\alpha_j^\varrho$ est donné par :

$$\overline{r}_j^\varrho = \psi^j - \sum_{k \in \Pi_j} \sum_{p \in \varrho \cap P^k} d^k lon_p^k(\mu), \tag{6.13}$$

## 6.2 Heuristique de construction de variables

Une des difficultés du modèle $FCM_{\mathbf{i}ar}$ réside dans le choix des arborescences à agréger (en d'autres termes quelles sont les sources des demandes que nous devrions combiner). Une première approche consiste à agréger de manière totalement aléatoire. Le premier risque d'une telle approche est qu'elle peut conduire à des tailles d'agrégation très différentes, et ainsi déséquilibrer les besoins en nombre de variables nécessaires pour chaque agrégation à la description de la solution optimale. Nous pourrions alors augmenter de manière significative le nombre d'itérations

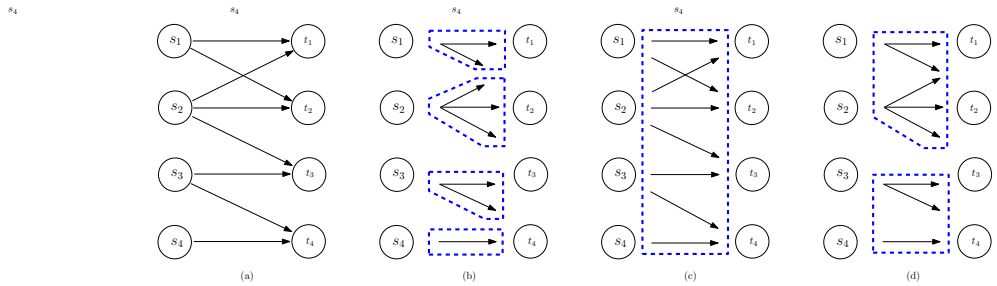



FIGURE 6.1 – *Plusieurs patrons d'agrégations représentés sur les graphes de demande : dans le graphe de demande (a), chaque arc représente une demande particulière (le modèle chemin étant basé sur cet ensemble K) ; dans (b), les demandes sont agrégées par source (modèle arborescent) ; dans (c), toutes les demandes sont agrégées en une seule ; dans (d), les arborescences de (b) sont agrégées deux par deux.*

globalement nécessaire pour trouver la solution optimale lors de la procédure de génération de colonnes, du fait de cette agrégation trop volumineuse. Une deuxième approche consiste à agréger de manière aléatoire ces sources, tout en contraignant ou en fixant la taille d'agrégation.

Nous proposons ici une heuristique qui va décider quelles sources (i.e. quelles variables d'arborescences) devraient être agrégées.

L'idée principale est de grouper différentes sources reliées par un chemin aller-retour de très grande capacité. Ces sources sont statistiquement voisines dans le graphe, et l'on peut s'attendre à ce que leurs demandes respectives soient acheminées de façon similaire. En d'autres termes, nous espérons qu'une variable globale $\alpha_j^o$ pour ces demandes utilise un grand nombre d'arcs similaires, ce qui aura pour effet de faire apparaitre cette variable dans un nombre restreint de contraintes de capacités 6.3.

L'objectif de l'algorithme est donc de créer un partitionnement $H = \Gamma(S)$ tel que chaque élément de $H$ correspond à un sous-ensemble de $S$ contenant au maximum $\mathbf{r}$ sources.

Pour chaque paire de sources $i$ et $j$, nous appelons $CapaciteMax(i,j)$, le chemin allant de $i$ à $j$ et qui possède une capacité maximale ; c'est-à-dire dont le minimum des capacités des arcs qu'il contient est maximal. Un algorithme classique ([3]) pour établir un tel chemin est explicité ci-après (6.2).

Pour chaque sous-ensemble (ou agrégat) $h \subset S$, nous définissons également une capacité interne $C(h)$, donnée par le minimum des $CapaciteMax(i,j)$ pour tout couple $i$ et $j$ de ce sous-ensemble. De ce fait $C(h) = \min_{i \in h, j \in h, j \neq i} CapaciteMax(i,j)$. Par convention, la capacité d'un singleton sera alors infinie.

L'algorithme crée d'abord un agrégat pour chaque source $h_s = \{s\}$. Le partitionnement $H$ contient dès lors $|S|$ sous-ensembles. Comme nous venons de le voir la capacité interne de ces singletons est alors établie comme infinie (étape 2 de l'algorithme). Une variable booléenne $Fusion$ permet de nous indiquer s'il est possible d'agréger deux sous-ensembles sans violer la contrainte de cardinalité donnée par $\mathbf{r}$. Nous supposerons initialement cette proposition vraie (étape 3). Ensuite les agrégats sont fusionnés itérativement tant que cette condition est vérifiée.

**Algorithm 13** Heuristique d'agrégation pour le modèle $FCM_{\mathbf{r}tr}$

**Require:** **r** cardinalité maximale pour chaque agrégat, $G(V, A)$ graphe orienté muni des capacités $c_a$

**Ensure:** Un partitionnement $H$ de l'ensemble des sources $S$

1: $H \leftarrow \{h_s = \{s\} : s \in S\}$ H contient $|S|$ sous-ensembles ; chaque sous-ensemble contenant exactement une source $s$
2: $C(h_s) \leftarrow \infty$ attribuer une capacité infinie pour $C(h_s)$, capacité de chaque agrégat (sous-ensemble) $h_s$
3: $Fusion \leftarrow true$ établir s'il est possible de fusionner deux agrégats sans violer la contrainte de cardinalité donnée par **r**
4: **while** $Fusion = true$ **do**
5: $MeillC \leftarrow -1$ Meilleur capacité courante
6: **for all** $h, h' \in H$, $h \neq h'$ **do**
7: $c \leftarrow \min\left\{C(h), C(h'), \min_{i \in h, j \in h'} CapaciteMax(i,j), \min_{i \in h', j \in h} CapaciteMax(i,j)\right\}$
8: **if** $c > MeillC$ et $|h'| + |h| \leq \mathbf{r}$ **then**
9: $MeillC \leftarrow c$ ; $h_1 \leftarrow h'$ ; $h_2 \leftarrow h$ garder en mémoire les meilleurs agrégats à fusionner, et la nouvelle capacité associée
10: **end if**
11: **end for**
12: **if** $MeillC > -1$ **then**
13: $h_1 \leftarrow h_1 \cup h_2$ ; $H \leftarrow H \setminus h_2$ ; $C(h_1) \leftarrow MeillC$ fusionner les deux agrégats et mettre à jour sa capacité interne (par défaut, l'agrégat "receveur" $h_1$)
14: **else**
15: $Fusion \leftarrow false$
16: **end if**
17: **end while**

Pour chaque combinaison de $h$ et $h'$ différents de notre partitionnement courant $H$, nous calculons $C(h \cup h')$ (étape 7). Nous utilisons les variables $MeillC$ pour stocker l'information de la meilleur capacité interne générée par l'agrégation de $h_1$ et $h_2$. Elle est de ce fait, initialisée à $-1$, avant la recherche de ce meilleur couple (étape 5). Ce meilleur couple $(h_1, h_2)$, déterminé en terme de capacité et qui satisfait la contrainte de cardinalité (vérifiée à l'étape 8) est alors fusionné en $h_1$, en mettant à jour sa capacité interne —minimum entre la capacité de $h_1$ de $h_2$ et des couples entre les deux ensembles—, ce qui produit un nouveau partitionnement $H$ (étape 13).

L'algorithme $CapaciteMax(i, j)$ ([3]),détermine de manière optimale le chemin de capacité maximale allant de $i$ à $j$.

L'algorithme 6.2 détermine le chemin de capacité maximale allant de $i$ à $j$. Pour ce faire, une arborescence des chemins de plus grande capacité partant de $i$ est construite itérativement, initalisée à l'ensemble vide (étape 1). Les capacités $C(j), (\forall j \in V)$, des chemins de capacité maximale sont initialisées à -1 (étape 2), excepté pour le nœud $i$ infinie (étape 3). Tant que le nœud $j$ ne se trouve pas dans cette arborescence, nous déterminons l'arc de capacité maximale reliant l'ensemble des nœuds de l'arborescence courante $\tau_i$, et son complémentaire. Nous utilisons alors $c_{\tilde{a}}$ la capacité initialisée à $-1$ (étape 5) et nous parcourons ces arcs (étape 6). Si une meilleure capacité de chemin $\min\{C(v^-(a')), c_{a'}\}$ est trouvée (étape 7), nous la stockons ainsi que l'arc associé (étapes 8 et 9). Ce meilleur arc $\tilde{a}$ est ajouté à l'arborescence courante $\tau_i$ (étape 9). Nous

**Algorithm 14** Algorithme de chemin de capacité maximale

**Require:** $i$ nœud de départ, $j$ nœud d'arrivée et $G(V, A)$ graphe orienté muni des capacités $c_a$
**Ensure:** Chemin $p^*$ de capacité maximale $c_{p^*}$

```
τ_i ← ∅ ensemble des arcs de l'arborescence courante
C(j) ← −1, ∀j ∈ V initialisation de la valeur des chemins de capacité maximale
C(i) ← ∞ capacité d'accès à i infinie
while j ∉ V^{τ_i} do
    c_ã ← −1 arc courant ã de capacité maximale entre τ_i et son complémentaire
    for all a' ∈ δ^+(V^{τ_i}) pour chaque arc sortant de l'ensemble des nœuds de l'arborescence courante do
        if min{C(v^−(a')), c_{a'}} > c_ã comparaison de la capacité par rapport à la meilleure trouvée then
            ã ← a'
            c_ã ← min{C(v^−(a')), c_{a'}}
        end if
    end for
    τ_i ← τ_i + ã mise à jour de l'arborescence τ_i
    C(v^+(ã)) ← c_ã mise à jour de la capacité pour l'accès au nœud v^+(ã)
end while
p^* ← p^{ij}_{τ_i} chemin à la capacité maximale allant de i à j
c_{p^*} ← min_{a∈p^*}{c_a} mesure de la capacité du chemin optimal
```

retournons enfin le chemin $p^*$ allant de $i$ à $j$ et qui possède une capacité maximale $c_{p^*}$.

**Proposition 23.** *L'algorithme 6.2 est exact et fortement polynomial, de complexité $O(|A|.|V|)$.*

*Démonstration.* (Rappel [3])

En utilisant l'algorithme 6.2 nous calculons itérativement le nœud dont l'accès possède le chemin de capacité la plus élevée.

Il détermine simplement :

$$\max_{p \in P^{ij}} \{\min_{a \in p} c_a\}, \tag{6.14}$$

que nous pouvons décomposer comme suit :

$$\max_{m \in V} \{\min\{\max_{p \in P^{im}} \{\min_{a \in p} c_a\}, \max_{p' \in P^{mj}} \{\min_{a \in p'} c_a\}\}\} \tag{6.15}$$

Si $C^*(i, j)$ est la valeur de capacité du chemin de capacité maximale allant de $i$ vers $j$, alors :

$$C^*(i, j) = \max_{m \in V} \{\min\{C^*(i, m), C^*(m, j)\}\} \tag{6.16}$$

Nous savons que nous pouvons décomposer le problème de la façon suivante, par rapport au nœud $i$ :

$$C^*(i, j) = \max_{a \in \delta^+(i)} \min\{c_a, C^*(v^+(a), j)\} \tag{6.17}$$

Nous savons également que :

$$C^*(i,j) \geq \min\{C^*(i,l), C^*(l,j)\}, \forall l \in V \tag{6.18}$$

Cela signifie qu'avec $i \neq j$ et $a^* = Arg(\max_{a \in \delta^+(i)} c_a)$

$$C^*(i, v^+(a^*)) = c_{a^*}, \tag{6.19}$$

que nous pouvons réécrire avec l'ensemble courant (arborescence) $V^{\tau_i} = \{i, v^+(a^*)\}$, et les arcs $\delta^+(\{i, v^+(a^*)\})$ divergents de cet ensemble :

$$C^*(i,j) = \max_{\tilde{a} \in \delta^+(\{i, v^+(a^*)\})} \min\{C^*(i, v^-(\tilde{a})), c_{\tilde{a}}, C^*(v^+(\tilde{a}), j)\}, \tag{6.20}$$

Nous venons donc de réduire le problème d'un nœud $v^+(a^*)$.

En établissant de manière générale $a^* = Arg(\max_{a \in \delta^+(V^{\tau_i})} \min\{c_{a^*}, C^*(i, v^-(a^*))\})$ , en réitérant de proche en proche, et établissant un nouveau $C^*(i, v^+(a^*))$ à chaque fois, nous pouvons généraliser les relations avec :

$$C^*(i,j) = \max_{a \in \delta^+(V^{\tau_i})} \min\{C^*(i, v^-(a)), c_a, C^*(v^+(a), j)\}, \tag{6.21}$$

ce qui pour le nœud $v^+(a^*)$ nous donne :

$$C^*(i, v^+(a^*)) = \min\{C^*(i, v^-(a^*)), c_{a^*}\}. \tag{6.22}$$

Cela signifie qu'après au plus $|A|$ opérations (recherche du meilleure arc $a^*$) nous avons réduit le problème jusqu'à ce que $v^+(a^*) = j$, ou plus généralement

$$\delta^+(V^{\tau_i}) = \emptyset$$

Nous construisons ainsi l'arborescence des chemins de capacités les plus grandes, en ajoutant un nœud à chaque fois. Nous répétons donc au maximum $|V|$ fois la procédure. Puisque chercher l'arc $a^*$, nécessite $|A|$ opérations, nous avons la complexité de notre algorithme en $O(|A|.|V|)$. □

Cet algorithme forme une heuristique d'agrégation produisant une formulation $FCM_{\mathbf{r}ar}$. Nous allons pouvoir exhiber des formulations meilleures que les formulations sus-nommées

$$FCM_{Chemin}$$

et

$$FCM_{Arbre}$$

.

Nous avons montré une manière statique d'agréger les sources, c'est-à-dire qu'à chaque variable on attribue un ensemble de demandes connues *a priori*. Une autre possibilité est de générer des variables agrégées de façon dynamique, c'est-à-dire de générer des agrégations de demandes décidées à chaque itération du problème maître.

Nous définissons alors les sous-ensembles possibles de $K$ par :

$$\Xi \subseteq K \tag{6.23}$$

Nous définissons alors un ensemble particulier $\xi$ :

$$\xi \in \Xi \tag{6.24}$$

$\xi$ est alors un ensemble variable, sous-ensemble de $K$. Soit $\Upsilon_\xi = \prod_{k\in\xi} P^k$ associé à $\xi$ l'ensemble des combinaisons de chemins possibles de $\xi$. On associe alors une variable $\zeta_\xi^\varrho$ à un sous ensemble $\xi$ et un ensemble de chemins associé à $\varrho$, lui-même vecteur de l'ensemble $\Upsilon_\xi = \prod_{k\in\xi} P^k$. Nous obtenons donc le modèle suivant :

$$(FCM_{Gener/Dyn}) \begin{cases} \max \gamma_{Gener/Dyn}, & & (6.25)\\ \displaystyle\sum_{\xi\in\Xi}\sum_{\varrho\in\Upsilon_\xi} \zeta_\xi^\varrho \sum_{k\in\xi} \sum_{\substack{p\in\varrho\cap P^k\\ p\ni a}} d^k \le c_a, & \forall a \in A, & (6.26)\\ \displaystyle\sum_{\substack{\xi\in\Xi\\ k\in\xi}}\sum_{\varrho\in\Upsilon_\xi} \zeta_\xi^\varrho \ge \gamma_{Gener/Dyn}, & \forall k \in K, & (6.27)\\ \zeta_\xi^\varrho \ge 0, & \forall \xi \in \Xi, \forall \varrho \in \Upsilon_\xi. & (6.28) \end{cases}$$

Sa version duale nous est alors donnée par :

$$\gamma^*_{Gener/Dyn} = \begin{cases} \min \displaystyle\sum_{a\in A} c_a\mu_a, & & (6.29)\\ \displaystyle\sum_{k\in K} \psi^k \ge 1, & & (6.30)\\ d^k lon^k_\varrho(\mu) \ge \psi^k & \forall \xi \in \Xi, \xi \ni k, \forall \varrho \in \Upsilon_\xi, & (6.31)\\ \mu_a, \psi^k \ge 0, & \forall a \in A, \forall k \in K, & (6.32) \end{cases}$$

où $lon^k_\varrho(\mu)$ représente la longueur du chemin pour la demande $k$, sur la structure $\varrho$, avec les poids $\mu_a$ sur les arcs. Rappelons alors que $(\mu_a)_{a\in A}$ and $(\psi^k)_{k\in K}$ sont les ensembles des variables duales associées respectivement aux contraintes (6.26) et (6.27).

Pour chaque demande, le coût réduit d'une variable $\zeta_\xi^\varrho$ sera exprimé de la manière suivante :

$$\overline{r}_\xi^\varrho = \sum_{k\in\xi} \psi^k - \sum_{k\in\xi} \sum_{p\in\varrho\cap P^k} d^k lon^k_p(\mu) \tag{6.33}$$

Là encore, étant donné un $\xi$ particulier, le coût réduit sera maximisé par le routage des demandes $k \in \xi$, sur les plus courts chemins selon les variables duales $(\mu_a)_{a\in A}$.

L'intérêt d'une telle formulation est que nous pouvons également décider de l'ensemble $\xi$, de sorte à maximiser également ce coût réduit au cours d'une seconde étape. Il s'agit tout simplement d'intégrer les demandes $k$ telles que :

$$\psi^k - \sum_{p\in\varrho\cap P^k} d^k lon^k_p(\mu) > 0 \tag{6.34}$$

Cette stratégie peut être choisie pour des agrégations particulières, comme l'agrégation d'arborescences : on peut alors générer une variable qui agrège toutes les arborescences de coût réduit strictement positifs. De la même manière, nous pouvons agréger les chemins de coûts strictement

positif, pour maximiser le coût réduit de l'arborescence (qui n'est alors pas nécessairement couvrante).

Cette méthode possède cependant une faiblesse importante : puisque les ensembles eux-mêmes sont variables, il est nécessaire de construire les variables de satisfaction de demande (6.27) pour l'ensemble le plus petit possible. Plus précisément, dans le cas d'agrégation d'arborescences, puisque nous ne pouvons préjuger des agrégations, il est nécessaire qu'une contrainte de demande soit explicitée pour chaque racine d'arborescence. De la même manière, dans le cas d'agrégation de chemins, une contrainte de satisfaction devra être explicitée pour chaque demande. Cela limite une partie du gain même de l'agrégation.

## Conclusion

Nous avons présenté ici un modèle général qui englobe ceux présentés précédemment, et permet d'en construire de nouveaux. Nous proposons ici une manière de construire un modèle en fonction de l'instance à résoudre, plus exactement nous nous appuyons sur l'efficacité du modèle arborescent et nous essayons de les combiner astucieusement, de manière à réduire le nombre de variables sans augmenter le nombre d'itérations. Le chapitre suivant a pour but de comparer ces modèles et d'en extraire des remarques empiriques.

# Chapitre 7

# Résultats numériques

## Introduction

Dans ce chapitre nous allons éprouver les performances des différents modèles présentés jusque-là, en étudiant deux types d'instances. Les premières sont des instances réputées difficiles pour le flot concurrent maximal, les secondes sont des instances générées aléatoirement. Nous allons pouvoir remarquer la pertinence du modèle arbre, ainsi que des modèles construits par l'heuristique d'agrégation.

## 7.1 Génération de graphes

Le générateur RMF, utilisé par [27, 8, 14], construit des instances de la manière suivante. Le graphe est constitué de $b$ trames. Une trame est constitué d'un carré de $a \times a$ nœuds. Chaque nœud est alors connecté par ses voisins directs (haut, bas, gauche et droite), quand ils existent (l'exception concerne les bords de la trame) par une capacité $a^2.C_{max}$. Les trames sont ensuite ordonnées, et sont connectées aux trames suivantes et précédentes (quand elles existent) par un matching parfait depuis les nœuds de la trame courante et celle ciblée, avec une capacité prise aléatoirement selon une loi uniforme dans l'intervalle $[C_{min}; C_{max}]$. De ce fait, les graphes possèdent exactement $ba^2$ nœuds et $6ba^2 - 4ab - 2a^2$ arcs.

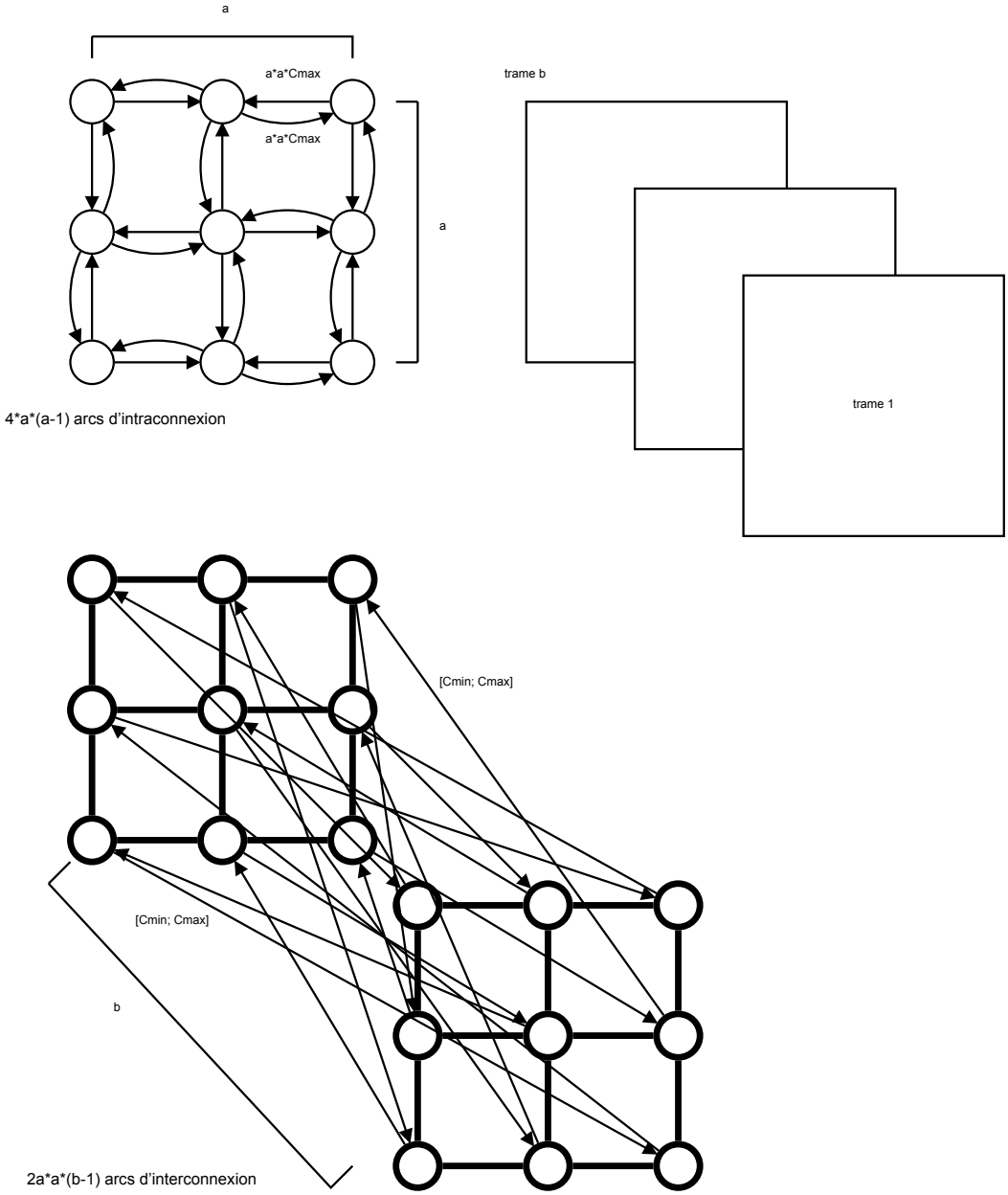



FIGURE 7.1 – *Types d'instances RMF selon les paramètres $a$, $b$ et $[C_{min}; C_{max}]$.*

Le générateur d'arcs aléatoires, part d'un état du graphe où seuls les nœuds sont présents, et ajoute un arc jusqu'à atteindre une densité fixée. Cela exclut les doublons (plus d'un arc reliant deux sommets), et les arcs liant un nœud sur lui-même.

## 7.2 Expérimentations

Toutes les expérimentations ont été exécutées sur un ordinateur doté d'un processeur Intel Core 2 Duo P8700 de fréquence 2.53 GHz, et de 3.48 Go de RAM. Le logiciel commercial Xpress (Mosel 3.20 - mmsystem 1.8.8 - mmxprs 2.2.0) été utilisé comme solveur de programmation linéaire.

Plusieurs expérimentations numériques ont été menées pour comparer les formulations chemins $FCM_{Chemin}$, arborescences $FCM_{Arbre}$, arc-flots agrégés $FCM_{Agg}$ et quelques cas particuliers de la formulation générique $FCM_{Gener}$.

Deux types de données ont été utilisées. Nous avons d'abord utilisé le générateur "RMFGEN" [27] pour générer des instances du FCM. Il est en effet admis par plusieurs auteurs que ces types d'instances sont parmi les plus difficiles pour ce problème. Les graphes "RMFGEN" sont caractérisés par deux paramètres $a$ et $b$, ainsi qu'un intervalle de capacité $[C_{min} = 5, C_{max} = 20]$.

Le second type de graphes est fourni par un générateur d'arcs aléatoires, dont les capacités sont également choisies dans l'intervalle précédent. Enfin, pour toutes les instances, les valeurs de demandes sont choisies de manière uniforme dans l'intervalle $[1, 10]$.

Puisque les formulations de $FCM_{Chemin}$, $FCM_{Arbre}$ et $FCM_{Gener}$ sont créées pour des procédures de génération de colonnes, la convergence peut être faible (c'est un effet classique de fin de queue). Beaucoup de techniques d'accélération de ces procédures ont été proposées dans la littérature (par exemple [12, 70] et les références contenues dans celles-ci). Dans notre implémentation nous avons utilisé la technique du In-Out, avec un $\epsilon$ identique aux paramètres standards de précision du solveur et un $\alpha = 0.25$ (ce qui correspond à 0.75 fois le point intérieur et 0.25 fois le point extérieur). L'intérêt ici n'est pas de comparer des techniques de stabilisation mais bien d'évaluer l'efficacité de différentes formulations du FCM.

Quatorze instances ont été créées. Les caractéristiques de ces instances sont synthétisées dans les tableaux 7.1 et 7.2. $I$ désigne simplement le numéro de l'instance.

Pour chaque instance le nombre de nœuds $|V|$, le nombre de sources $|S|$, le nombre de destinations pour chaque source $|T(s)|$ et le nombre d'arcs $|A|$ sont fournis dans les tableaux 7.1 et 7.2. Les valeurs de $a$ et $b$ sont également données dans le tableau 7.1 relatif aux instances RMF (générées par le programme "RMFGEN").

| $I$ | $\|V\|$ | $\|S\|$ | $\|T(s)\|$ | $a$ | $b$ | $\|A\|$ |
|---|---|---|---|---|---|---|
| 1 | 125 | 125 | 31 | 5 | 5 | 600 |
| 2 | 125 | 63 | 62 | 5 | 5 | 600 |
| 3 | 125 | 31 | 124 | 5 | 5 | 600 |
| 4 | 125 | 125 | 124 | 5 | 5 | 600 |
| 5 | 216 | 216 | 215 | 6 | 6 | 1080 |
| 6 | 343 | 343 | 342 | 7 | 7 | 1764 |
| 7 | 512 | 512 | 511 | 8 | 8 | 2688 |
| 8 | 125000 | 1 | 124999 | 50 | 50 | 735000 |

TABLE 7.1 – Caractéristiques des huit instances RMF.

| $I$ | $\|V\|$ | $\|S\|$ | $\|T(s)\|$ | $\|A\|$ |
|---|---|---|---|---|
| 9 | 180 | 180 | 179 | 2000 |
| 10 | 180 | 180 | 179 | 2500 |
| 11 | 180 | 180 | 179 | 3000 |
| 12 | 250 | 250 | 249 | 2000 |
| 13 | 250 | 250 | 249 | 2500 |
| 14 | 250 | 250 | 249 | 3000 |

TABLE 7.2 – Caractéristiques des six instances de graphe aléatoires.

L'évaluation des performance des formulations FCM est basée en grande partie sur le temps de calcul requis pour l'obtention de la solution optimale. Cependant, nous avons également consigné le nombre de variables générées durant la procédure ("Gen. Vars") et le nombre de variables non nulles lors de la dernière itération, qui décrivent de ce fait la solution optimale ("Opt. Vars") et le nombre d'itérations, c'est-à-dire le nombre de fois que le problème maître a été résolu ("Iterations"). Lorsque la formulation ne nous a pas permis d'obtenir la solution optimale (à cause d'un manque mémoire) le symbole "-" est placé à la place du nombre.

Il est important de souligner et de rappeler que l'algorithme de génération de colonne proprement dit est exactement le même dans chacune des formulations : il s'agit de trouver une arborescence des plus courts chemins depuis chaque source.

Dans le tableau 7.3, nous comparons les formulations $FCM_{Agg}$, $FCM_{Chemin}$, $FCM_{Arbre}$ et $FCM_{Tout}$, en utilisant les instances RMF, tandis que le tableau 7.4 correspond à des instances de graphes aléatoires.

| | CPU (s) | | | | Gen. Vars | | |
|---|---|---|---|---|---|---|---|
| $I$ | Agg | Chemin | Arbre | Tout | Chemin | Arbre | Tout |
| 1 | 146 | 32 | 20 | 326 | 8309 | 1273 | 338 |
| 2 | 62 | 25 | 11 | 163 | 9252 | 798 | 287 |
| 3 | 31 | 20 | 9 | 54 | 9464 | 534 | 179 |
| 4 | 229 | 433 | 19 | 234 | 38108 | 1376 | 263 |
| 5 | 3162 | 26927 | 341 | 3171 | 126389 | 3615 | 598 |
| 6 | 28430 | - | 1409 | - | - | 5367 | - |
| 7 | 310590 | - | 3526 | - | - | 7621 | - |

| | Opt. Vars | | | Ite érations | | |
|---|---|---|---|---|---|---|
| $I$ | Chemin | Arbre | Tout | Chemin | Arbre | Tout |
| 1 | 3964 | 206 | 85 | 42 | 18 | 338 |
| 2 | 3972 | 125 | 67 | 39 | 21 | 287 |
| 3 | 3964 | 96 | 41 | 32 | 32 | 179 |
| 4 | 15693 | 190 | 73 | 45 | 14 | 263 |
| 5 | 46855 | 386 | 150 | 56 | 29 | 598 |
| 6 | - | 544 | - | - | 22 | - |
| 7 | - | 858 | - | - | 23 | - |

TABLE 7.3 – Comparaison des formulations $FCM_{Agg}$, $FCM_{Chemin}$, $FCM_{Arbre}$ et $FCM_{Tout}$ : instances RMF

| | CPU (s) | | | | Gen. Vars | | |
|---|---|---|---|---|---|---|---|
| $I$ | Agg | Chemin | Arbre | Tout | Chemin | Arbre | Tout |
| 9 | 20537 | 263 | 39 | 1310 | 65150 | 1973 | 343 |
| 10 | 32347 | 1257 | 89 | 4876 | 71709 | 2655 | 669 |
| 11 | 85980 | 3036 | 515 | - | 80944 | 3577 | - |
| 12 | 16174 | 10002 | 220 | - | 160210 | 3325 | - |
| 13 | 30973 | 32380 | 323 | - | 193545 | 3675 | - |
| 14 | 172018 | 47294 | 1545 | - | 241971 | 4856 | - |

| | Opt. Vars | | | Itérations | | |
|---|---|---|---|---|---|---|
| $I$ | Chemin | Arbre | Tout | Chemin | Arbre | Tout |
| 9 | 32560 | 257 | 100 | 31 | 13 | 343 |
| 10 | 32689 | 341 | 172 | 50 | 17 | 669 |
| 11 | 33131 | 775 | - | 52 | 24 | - |
| 12 | 62731 | 588 | - | 32 | 15 | - |
| 13 | 63120 | 484 | - | 32 | 19 | - |
| 14 | 62903 | 1103 | - | 46 | 24 | - |

TABLE 7.4 – Comparaison des formulations $FCM_{Agg}$, $FCM_{Chemin}$, $FCM_{Arbre}$ et $FCM_{Tout}$ : instances de graphes aléatoires

| | CPU (s) | | | Gen. Vars | | | Opt. Vars | | | Itérations | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $I$ | Arbre | $\mathbf{2}Ar$ | $\mathbf{4}Ar$ | Arbre | $\mathbf{2}Ar$ | $\mathbf{4}Ar$ | Arbre | $\mathbf{2}Ar$ | $\mathbf{4}Ar$ | Arbre | $\mathbf{2}Ar$ | $\mathbf{4}Ar$ |
| 1 | 20 | 18 | 28 | 1273 | 947 | 794 | 206 | 153 | 120 | 18 | 19 | 29 |
| 2 | 11 | 11 | 15 | 798 | 549 | 450 | 96 | 108 | 81 | 21 | 23 | 32 |
| 3 | 9 | 13 | 29 | 534 | 459 | 394 | 96 | 81 | 68 | 32 | 40 | 53 |
| 4 | 19 | 18 | 26 | 1376 | 905 | 719 | 190 | 127 | 95 | 14 | 17 | 25 |
| 5 | 341 | 311 | 360 | 3615 | 2502 | 1955 | 386 | 290 | 222 | 29 | 28 | 42 |
| 6 | 1409 | 1301 | 1444 | 5367 | 3856 | 2444 | 544 | 347 | 254 | 22 | 28 | 32 |
| 7 | 3526 | 2701 | 3453 | 7621 | 4501 | 3524 | 858 | 573 | 424 | 23 | 23 | 33 |

TABLE 7.5 – Comparaison des formulations $FCM_{Arbre}$, $FCM_{\mathbf{2}Ar}$ et $FCM_{\mathbf{4}Ar}$ : Instances RMF

Si nous nous concentrons sur les résultats liés aux instances 1 à 3, nous pouvons remarquer que la résolution basée sur la formulation arborescente est environ deux fois plus rapide que la formulation chemin. Cela semble lié au nombre de variables générées qui est bien plus faible dans la première formulation. Cependant, même si le nombre de variables générées dans la formulation $FCM_{Tout}$ est encore plus faible. Cette dernière semble néanmoins la moins efficace. Cela est dû au nombre d'itérations nécessaires pour atteindre la solution optimale, extrêmement grand par rapport à ceux des autres formulations.

Pour ce qui est des instances 2 et 3, le nombre de sources diminue en même temps que le nombre de destinations par source augmente, gardant approximativement constant le nombre de demandes ($|K|$ presque constant pour les instances 1, 2 et 3). Il s'avère que cette réduction de sources en gardant le même nombre de demandes profite beaucoup à la formulation arborescente comparée à celle chemin, avec un temps de calcul près de deux fois inférieur pour l'instance 3. Ce comportement s'explique par le fait que la formulation arborescente agrège l'information

| | CPU (s) | | | Gen. Vars | | | Opt. Vars | | | Itérations | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $I$ | Arbre | $\mathbf{2}Ar$ | $\mathbf{4}Ar$ | Arbre | $\mathbf{2}Ar$ | $\mathbf{4}Ar$ | Arbre | $\mathbf{2}Ar$ | $\mathbf{4}Ar$ | Arbre | $\mathbf{2}Ar$ | $\mathbf{4}Ar$ |
| 9 | 39 | 29 | 42 | 1973 | 1071 | 759 | 257 | 188 | 100 | 13 | 12 | 17 |
| 10 | 89 | 72 | 83 | 2655 | 1529 | 1010 | 341 | 258 | 187 | 17 | 18 | 23 |
| 11 | 515 | 429 | 819 | 3577 | 2456 | 1950 | 775 | 647 | 504 | 24 | 27 | 44 |
| 12 | 220 | 180 | 314 | 3325 | 2181 | 1560 | 588 | 389 | 389 | 15 | 18 | 25 |
| 13 | 475 | 460 | 633 | 3699 | 2676 | 1888 | 766 | 595 | 509 | 17 | 22 | 30 |
| 14 | 1545 | 1153 | 2844 | 4856 | 3309 | 2498 | 1103 | 856 | 767 | 24 | 28 | 40 |

TABLE 7.6 – Comparaison des formulations $FCM_{Arbre}$, $FCM_{\mathbf{2}Ar}$ et $FCM_{\mathbf{4}tr}$ : instances de graphes aléatoires

liée à toutes ses destinations ; cette agrégation est d'autant plus importante que le nombre de destinations par source est grand. C'est également un phénomène que l'on retrouve logiquement par la formulation arc-flot, pour des raisons similaires.

Pour l'instance 4, le nombre de sources $|S|$ est plus grand que pour l'instance 3, alors que le nombre de destinations par source reste constant. Nous pouvons observer alors que la résolution avec la formulation arborescente devient environ vingt fois plus rapide que la formulation chemin.

Si nous regardons ce comportement sur des instances encore plus grandes (5-6-7), la différence croît significativement. La formulation chemin devient d'ailleurs incapable de résoudre ces instances du fait de l'insuffisance en mémoire vive. On remarque alors que la formulation arborescente devient plus de 88 fois plus rapide que la formulation arc-flots agrégée pour l'instance 7. il est également important de souligner que l'efficacité des arborescences se mesure également en termes d'itérations puisqu'elle en nécessite moins que pour les chemins.

On constate que la formulation d'agrégation totale $FCM_{Tout}$ devient elle-même plus efficace que la formulation chemin pour les instances 4 et 5.

Des conclusions équivalentes sont obtenues lorsque le seconde type d'instances est testé. Le tableau 7.4 montre très clairement que la formulation arborescente surpasse les autres formulations.

La différence principale entre les résultats du tableau 7.3 et ceux du tableau 7.4 semble être la performance de $FCM_{Chemin}$ par rapport à $FCM_{Aggr}$. Dans le cas RMF, du moins pour les grandes instances, $FCM_{Chemin}$ est plus difficile à résoudre que son homologue arc-flots, alors que la formulation chemin est plus efficace que $FCM_{Aggr}$ sur les graphes aléatoires. Quoiqu'il en soit, la formulation arborescente reste la plus efficace dans tous les cas.

Cette dernière est alors comparée à des formulations génériques dans les tableaux 7.5 et 7.6. Nous avons expérimenté deux types d'agrégation : une 2-arborescence $FCM_{\mathbf{2}Ar}$ et une 4-arborescence $FCM_{\mathbf{4}Ar}$ (une variable représente 4 arborescences de 4 racines différentes correspondant à des sources différentes), lorsque l'heuristique a été utilisée.

On peut observer que le nombre d'itérations relatif à $\mathbf{2}Ar$ est généralement plus élevé que celui de la formulation arborescente mais le nombre de variables générées est aussi plus faible. On peut décrire de façon grossière la tendance suivante : plus l'on agrège, moins il y a de variables générées, mais plus le besoin en nombre d'itérations grandit. Par ailleurs, $FCM_{\mathbf{2}Ar}$ peut s'avérer

plus efficace que la formulation arborescente. Cela signifie qu'agréger des demandes par source et considérer des arborescences à la place des chemins est une bonne stratégie, mais elle n'est pas nécessairement la meilleure. La meilleure stratégie d'agrégation résulte d'un compromis décrit par la tendance précédente.

Nous présentons enfin dans la tableau 7.7 les résultats de $FCM_{Chemin}$ et $FCM_{Arbre}$, pour les instances 1 à 5 lorsque la procédure In-Out n'est pas appliquée. Nous nous assurons dès lors que la performance de la formulation arborescente par rapport à la formulation chemin n'est pas seulement liée à l'utilisation de cette méthode. Le tableau montre très clairement que $FCM_{Arbre}$ reste bien meilleur que $FCM_{Chemin}$. De plus, si nous comparons ces résultats avec les précédents (avec In-Out) nous pouvons voir le grand gain en temps de calcul que nous pouvons obtenir grâce à cette technique. Il est intéressant de noter que cette technique est plus profitable pour la formulation arborescente que pour la formulation chemin. Ceci peut s'expliquer par le fait que l'arborescence étant un support plus volumineux d'information, sa caractérisation est plus cruciale ; optimiser le choix de sa génération atténue sa principale faiblesse, à savoir son manque de souplesse.

| | CPU time (s) | | Gen. Vars | | Opt. Vars | | Itérations | |
|---|---|---|---|---|---|---|---|---|
| $I$ | Chemin | Arbre | Chemin | Arbre | Chemin | Arbre | Chemin | Arbre |
| 1 | 251 | 107 | 33365 | 5375 | 4016 | 246 | 116 | 89 |
| 2 | 333 | 83 | 39818 | 3588 | 4047 | 188 | 113 | 95 |
| 3 | 353 | 75 | 40010 | 2031 | 3988 | 142 | 104 | 106 |
| 4 | 5195 | 451 | 116243 | 8655 | 15747 | 254 | 99 | 103 |
| 5 | 95205 | 4569 | 362838 | 24013 | 46870 | 438 | 129 | 157 |

TABLE 7.7 – Comparaison entre les formulations chemin et arborescence lorsque la procédure In-Out n'est pas utilisée

# Conclusion

La pertinence d'une formulation de multi-flots semble liée à des phénomènes antagonistes au niveau de la rapidité de résolution. L'augmentation de l'agrégation d'informations au niveau des variables permet de diminuer le nombre de variables à générer, et donc la résolution d'une itération du problème maître. Cependant, agréger de l'information peut induire une "granularité" trop épaisse, et donc un nombre de variables nécessaire pour décrire la solution optimale plus grand. Dans le cadre du flot concurrent maximal, on peut remarquer que de ces effets résulte une diminution des variables nécessaires à la résolution exacte. S'il s'agissait de l'unique levier d'efficacité des modèles, l'agrégation maximale serait la stratégie la plus intéressante. Dans la pratique on remarque que l'agrégation diminue la capacité du modèle à converger rapidement (plus d'itérations nécessaires).

Ceci peut s'expliquer à nouveau par la granularité proposée par le modèle. La construction d'une variable agrégée nécessite implicitement plus de choix de construction qui resteront figés par la suite. La pertinence d'une variable agrégée est plus étroitement liée à une configuration donnée et elle a donc moins de souplesse pour "s'adapter" à une solution très différente (variables apparaissant dans beaucoup de contraintes/ densité élevée de la matrice de contrainte).

Pour corriger en partie ce comportement pathologique, nous avons fait appel à une technique de stabilisation existante : le in-out. Le but de cet algorithme est de construire des variables plus proches de ce qui définit le polyèdre du problème, et donc d'améliorer la rapidité de convergence. On remarque alors que cette méthode est d'autant plus pertinente que la formulation est agrégée.

En pratique, si les formulation arborescentes semblent être un bon compromis entre le nombre de variables et le nombre d'itérations nécessaires, on peut voir qu'il est possible de construire des modèles encore plus adaptés, qui s'inspirent directement de la structure de l'instance considérée. On peut donc trouver un nombre d'agrégations intéressant, qui peut diffèrer avec la taille même de l'instance. Tout comme il est préférable d'utiliser une formulation chemin qu'une formulation explicite (arc-flot par exemple) lorsque le graphe grandit, on remarque que ce nombre "optimal" d'agrégation semble également augmenter avec la taille de l'instance. Il serait intéressant d'étudier plus en profondeur ce type de comportement.

Nous allons maintenant nous concentrer sur une approche de résolution différente pour traiter un cas particulier : le flot concurrent maximal dans le cas mono-source.

# Chapitre 8

# Le problème de flot concurrent maximal, cas mono-source

## Introduction

Ce chapitre traite le flot concurrent maximal dans sa version où il n'existe qu'une seule source (mais plusieurs destinations, aux demandes différentes). Ce problème est un problème régulièrement rencontré dans le domaine des télécommunications, et dans notre contexte modélise particulièrement bien le paradigme de réseau de distribution de contenus —un serveur central, et des clients à satisfaire. De plus, dans le cadre de plusieurs serveurs distincts, la possibilité du choix du serveur émetteur du trafic pour satisfaire un client peut se ramener à nouveau à ce problème par une modification triviale. Nous rappelons d'abord les algorithmes polynomiaux pour le problème particulier de flot maximum ainsi que leurs spécificités. Nous introduisons notre nouvel algorithme combinatoire pour résoudre le maximum concurrent flot mono-source et prouvons ensuite l'exactitude et la forte polynomialité de cette approche. Nous concluons ce chapitre par une démonstration de ses performances par rapport à des approches de programmation linéaire, et notamment l'un des meilleurs modèles que nous avons présenté : le modèle arbre.

## 8.1 Algorithmes de Ford-Fulkerson et Edmonds-Karp, problème de flot maximum

Le problème de flot maximum est un grand classique de la littérature. Parmi les premiers à l'avoir étudié on nomme Ford et Fulkerson. Le problème consiste à envoyer un flot maximum de la source (unique) vers la destination (également unique), sur un graphe orienté, dont les arcs sont munis de capacités. C'est donc un cas particulier des modèles FCM que nous avons vu précédemment.

**Définition 30.** *Un flot $F, f(a) \forall a \in A$ de $s$ à $t$, sur le graphe $G = (V, A)$ est réalisable si et seulement si :*

1. $f(a) \leq c_a, \forall a \in A$
2. $\sum_{j \in \delta^+(i)} f(a = (ij)) = \sum_{j \in \delta^-(i)} f(a = (ij)), \forall i \in V - \{s, t\}$

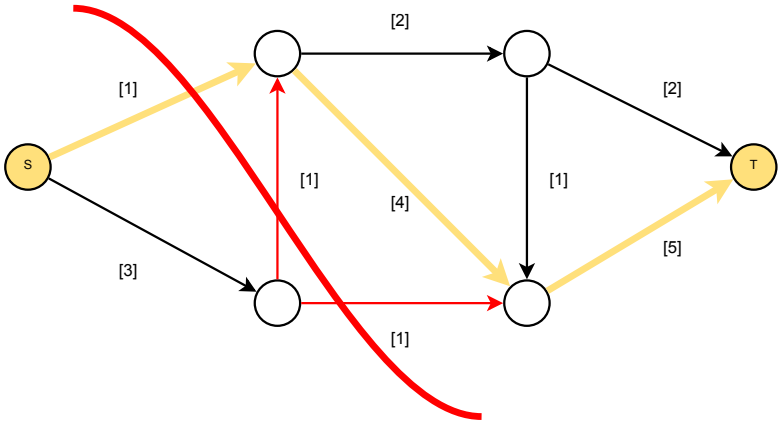



FIGURE 8.1 – *Problème du flot maximum (resp. son dual : coupe minimale) sur un exemple de graphe. Le but est de transmettre le flot maximum de la source $S$ vers la destination $T$, sous contraintes de capacité sur chaque arc (indiquées ici entre crochets).*

Une approche consiste à créer itérativement un chemin de flots augmentant depuis la source vers la destination. L'exemple de la figure 8.2 nous montre un tel principe, ainsi que le cas d'arrêt. Le chemin augmentant choisi (en rouge) est le support du flot augmentant. Il est limité par une capacité de 2. Une fois ce flot effectivement acheminé de la source vers la destination, aucun chemin de flot non nul ne peut plus atteindre la destination sur le graphe $G = (V, A)$. Pourtant la solution n'est pas optimale.

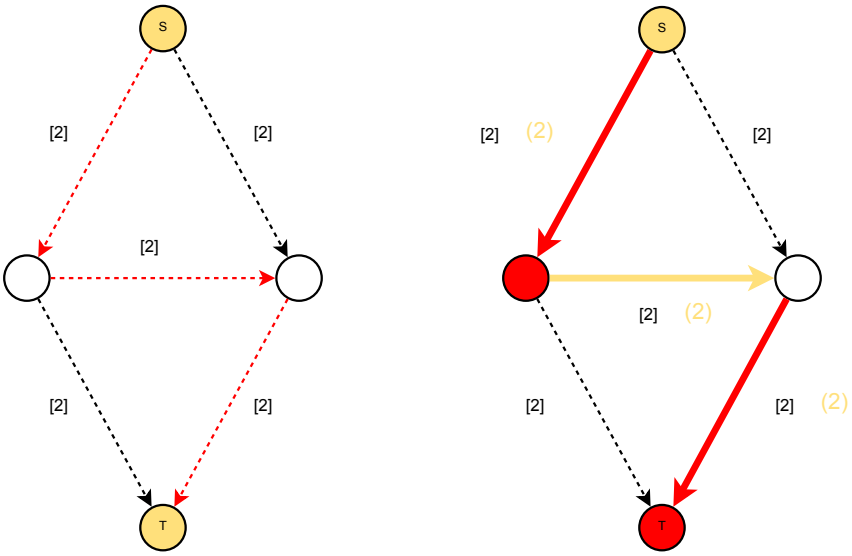
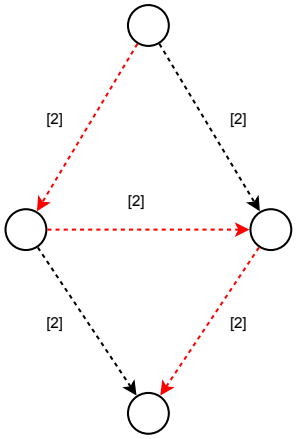



FIGURE 8.2 – *Non optimalité des flots augmentant sur le graphe "classique".*

Ford et Fulkerson proposent alors un algorithme combinatoire pour résoudre ce problème en introduisant la notion de graphe d'écart. Tout comme l'algorithme du simplexe, où des variables

d'écart sont introduites ; les auteurs ajoutent ici un arc "de retour" associé à un arc dans le sens opposé, de capacité égale à la valeur de flot qui transite sur cet arc initial. De ce fait à chaque instant, la somme des capacités d'un arc initial et de son arc de retour associé sera constante. La capacité d'un arc de retour sera utilisable par un chemin augmentant au même titre qu'un arc initialement décrit par $A$. Il est alors intéressant de noter que cela permet toujours d'assurer une solution réalisable, autant au niveau des contraintes de capacités qu'au niveau des contraintes de conservation de flots.

L'exemple des figures 8.3 et 8.4 illustrent le mécanisme de graphe d'écart, et l'utilisation d'un arc de retour. Le chemin choisi en rouge sur le graphe initial (a), va modifier les capacité du graphe d'écart. En mettant à jour le flot augmentant de 2, on obtient les deux graphes (classique et d'écart) ci-après (b). Si dans le graphe classique, il n'est plus possible de créer un flot augmentant, on peut tout à fait le faire dans le graphe d'écart (noté en rouge). En faisant passer un flot augmentant de 1, nous obtenons les deux graphes correspondant qui suivent (c).




(a)

FIGURE 8.3 – *Graphe initial.*

**Définition 31.** *Le graphe d'écart $\tilde{G} = (V, \tilde{A})$ induit par le graphe $G = (V, A)$ et un flot réalisable $f(a), \forall a \in A$ sur ce graphe est définit par :*

1. *L'ensemble des nœuds $V$ identique à $G$*
2. *Chaque arc $a = (ij) \in A$ muni de la capacité $c_{\tilde{a}} = c_a - f(a)$*
3. *Chaque arc $a' = (ji)/a = (ij) \in A$ muni de la capacité $c_{\tilde{a}} = f(a)$*

**Définition 32.** *Un flot augmentant de $\tilde{G}$ est un quantité $\mu$ sur chemin $p$ de $s$ vers $t$, tel que :*

$$f(a) + \mu \leq c_{\tilde{a}}, \forall a \in p$$

*C'est une modification légale de flot, puisqu'elle respecte nécessairement les deux conditions de faisabilité.*

**Définition 33.** *Soit un graphe $G = (V, A)$, avec des sommets $s$ et $t$ particuliers désignant respectivement la source et la destination. Chaque arc $a \in A$ est muni d'une capacité $c_a$. Le problème de coupe minimal consiste à trouver l'ensemble d'arcs $A'$ ($A' \subset A$) partitionnant $V$ en deux ensembles $S$ —contenant le nœud $s$ (source), et $T$ —contenant le nœud $t$ (destination), tel*

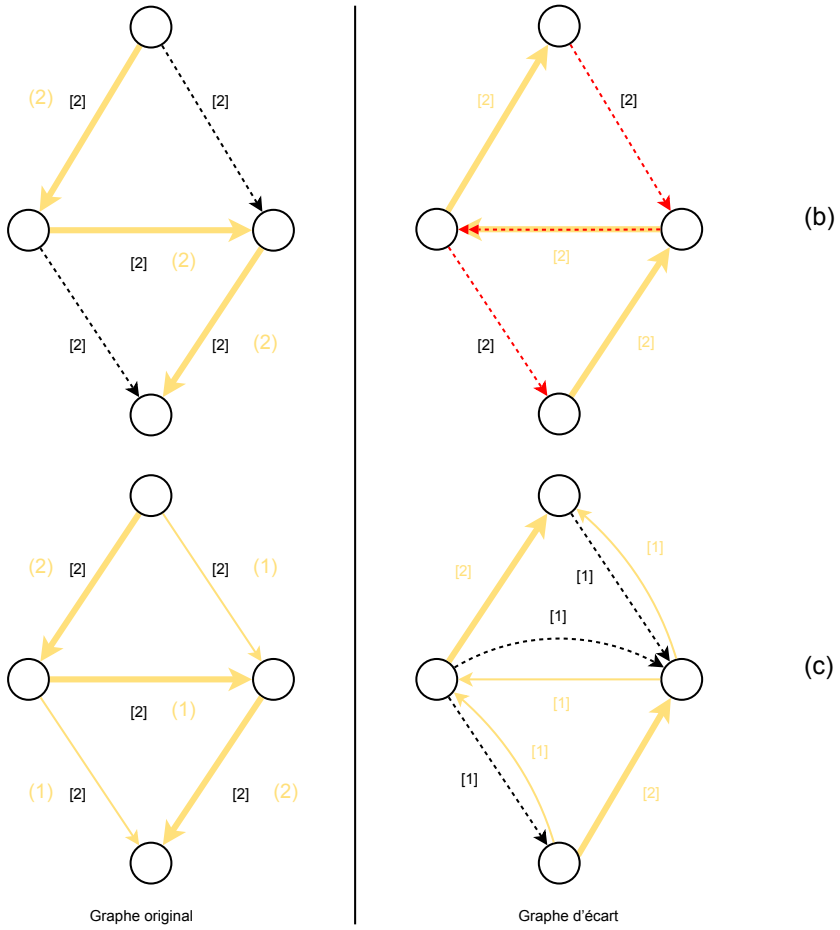



FIGURE 8.4 – *Principe du graphe d'écart.*

*que la somme des capacités* $C(A') = C(S,T)$ *des arcs dont l'origine est contenue dans* $S$ *et la destination dans* $T$ *est minimum :*

$$\min_{S,T/s\in S, t\in T, S\cup T=V, S\cap T=\emptyset} C(S,T)$$

**Théorème 24.** *Soit un graphe* $G = (V, A)$ *avec des sommets* $s$ *et* $t$ *particuliers désignant respectivement la source et la destination. Chaque arc* $a \in A$ *est muni d'une capacité* $c_a$*. La valeur optimale du flot maximal et de la valeur de la coupe minimale sont identiques.*

*Démonstration.* La preuve découle du théorème de dualité forte, le problème de flot maximum étant le problème dual de problème de coupe minimale (et inversement). □

Ford et Fulkerson prouvent alors l'algorithme exact en cas d'arrêt, par l'égalité avec une solution réalisable duale : la coupe minimale. Les figures 8.5 et 8.6 représentent l'amélioration successive du flot sur le graphe d'écart. Par rapport à l'exemple précédent 8.4, nous pouvons ajouter un autre chemin de flots augmentant **de valeur maximale** ; la solution passe de $\gamma = 2$ à $\gamma = 4$. Nous obtenons alors une valeur du problème primal égale à la coupe minimale (valeur réalisable du problème dual), représentée ici en rouge.

A la dernière étape, il est impossible de trouver un flot augmentant : nous obtenons la coupe minimale qui nous garantit que notre solution est alors optimale.

**Algorithm 15** : flot maximum - Edmonds-Karp

**Require:** un graphe orienté $G = (V, A)$ muni des capacités $(c_a)_{a \in A}$
**Ensure:** la valeur optimale $\gamma^*$ du flot maximum de $s$ vers $t$.

1: Initialisation de $\gamma^* \leftarrow 0$.
2: **while** Il existe un flot augmentant non nul sur le graphe d'écart **do**
3: Trouver le $\gamma$ maximum tel que le flot $\gamma$ peut être acheminé sur le chemin augmentant entre $s$ et $t$ sans violer de contraintes de capacité.
4: Augmenter le(s) flot(s) selon $\gamma$, et mettre à jour le graphe
5: $\gamma^* \leftarrow \gamma^* + \gamma$
6: **end while**

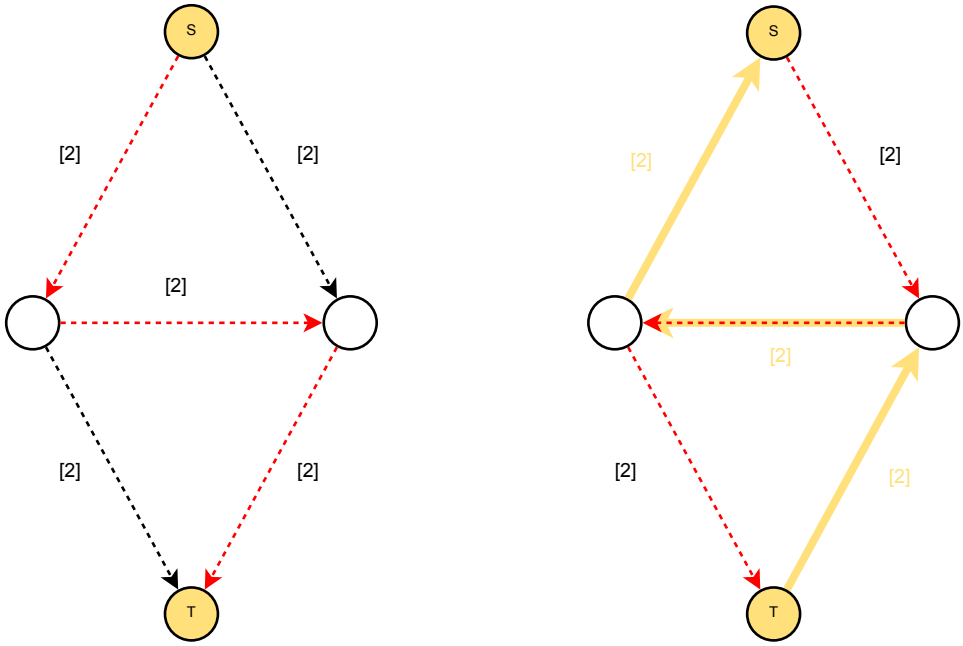



FIGURE 8.5 – *Première et seconde itération...*

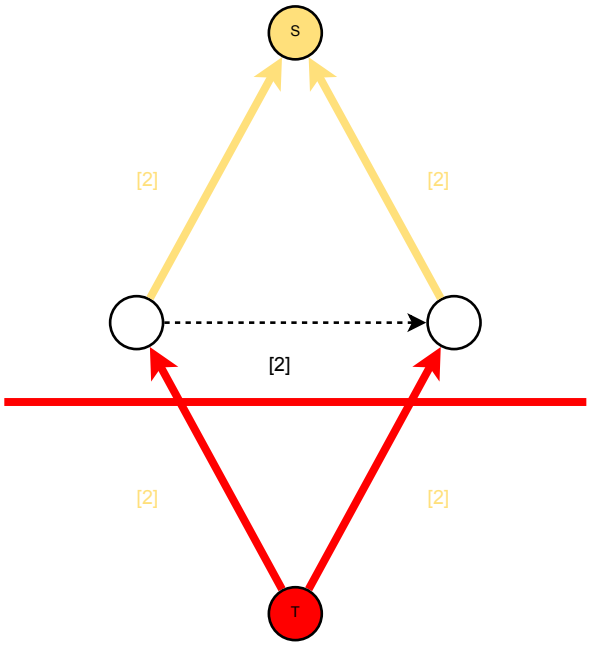



FIGURE 8.6 – *Solution optimale obtenue.*

Les failles de cet algorithme seront montrées plus tard, à l'aide d'un exemple où un nombre infini d'opérations peut être effectué, et qui ne converge pas même vers l'optimum. Edmonds et Karp modifient alors l'algorithme en imposant que le chemin de flots augmentant soit choisi pour être un plus court chemin légal du graphe d'écart. Muni de cette propriété, la polynomialité (complexité en $O(|V|.|A|^2)$) de l'algorithme peut alors être prouvée.

D'autres algorithmes existent de même complexité comme le "Push and relabel" [4] ou encore le "blocking flow algorithm" de [29]. Des modifications dans le fonctionnement de l'algorithme peuvent permettre d'obtenir des complexités particulières (voir tableau 8.1).

## 8.2 Formulation FCM mono-source et principe de l'algorithme

Nous allons proposer ici un algorithme fortement polynomial (voir définition ci-dessous et [46]) pour le cas où les demandes partagent toutes une unique source. Nous supposerons dès lors que les demandes $k \in K$ ont donc toutes pour source $s = s^k, \forall k \in K$.

**Définition 34.** *Un algorithme est dit fortement polynomial s'il satisfait les deux conditions suivantes :*

*(i)- Le nombre d'opérations élémentaires nécessaires à son exécution est borné par une fonction polynomiale des paramètres d'entrée du problème*

*(ii)- L'espace mémoire nécessaire à son fonctionnement est borné par une fonction polynomiale des paramètres d'entrée du problème*

| Nom | Description | Complexité |
|---|---|---|
| Ford Fulkerson | Réitération de flot augmentant de $s$ à $t$ sur un graphe d'écart jusqu'à impossibilité | $O(A.max\|f\|)$ (poids entiers uniquement) |
| Edmonds Karp | Comme Ford Fulkerson, sur des plus courts chemins (en termes d'arcs) | $O(A.V^2)$ |
| Dinitz "blocking flow algorithm" | réitération de flots bloquants dans un graphe pondéré aux nœuds | $O(A.V^2)$ |
| "Push and relabel algorithm" | Ajout itératif de préflots "push" (fonction de flots avec possibilité d'excès sur les nœuds) et mise à jour d'étiquettes de poids sur les arcs "relabel" | $O(A.V^2)$ |
| "Push and relabel algorithm" avec une liste FIFO | Sélection des nœuds selon une liste FIFO | $O(V^3)$ |
| Dinitz "blocking flow algorithm" avec structure de donnée en arbres | La structure de données en arbres améliore l'exécution du flot maximum sur le graphe pondéré | $O(\|V\|.\|A\| .log(\|V\|)$ |
| "Push and relabel algorithm" avec structure de donnée en arbres | La structure de données en arbres améliore l'exécution du flot maximum sur le graphe pondéré | $O(\|V\|.\|A\| .log(\|V\|^2/\|A\|)$ |
| MPM (Malhotra, Pradmodh-Kumar et Maheshwari) algorithm | | $O(V^3)$ |
| Algorithme d'Orlin | | $O(\|V\|.\|A\|)$ |

TABLE 8.1 – Principaux algorithmes polynomiaux pour le problème de flot maximum.

Une approche possible pour résoudre le FCM dans sa version mono-source est de ramener ce problème à un problème de flot maximum, dont on connaît l'algorithme polynomial pour le résoudre. Mais cette transformation nécessite de placer des capacités variables sur les arcs additionnels, elles-même dépendantes du flot maximum, ce sont en réalité des contraintes d'équilibrage ("Load Balancing") qui sont ici appliquées (8.7 et 8.8). Dès lors, il est impossible d'utiliser l'algorithme d'Edmonds-Karp sur ce graphe, mais des recherches dichotomiques peuvent être utilisées. Ce type de procédure ne génère généralement que des algorithmes d'approximation.

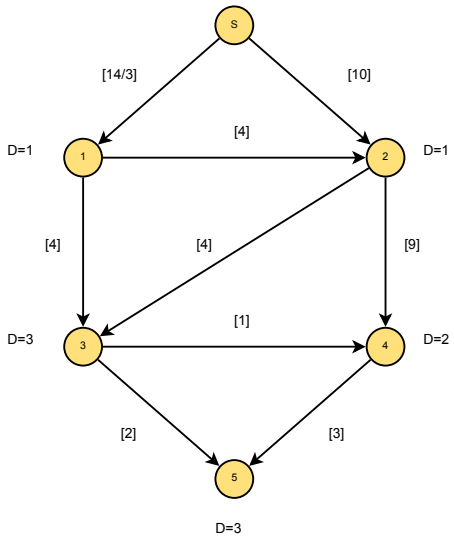



FIGURE 8.7 – *Equivalence du flot concurrent maximum... (cas mono-source).*

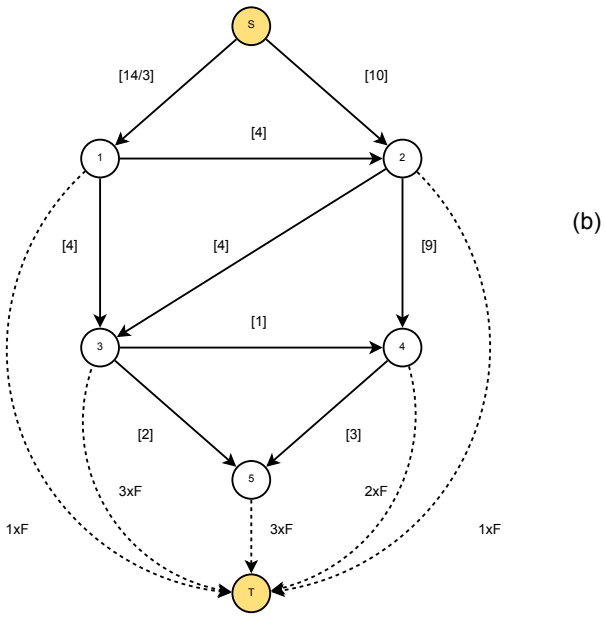



FIGURE 8.8 – *...en flot maximum simple (cas mono-source).*

Sur les figures 8.7 (a), le graphe avec la source $S$ et les différentes destinations munies de leurs valeurs. supposons que nous connaissons $F$ la valeur du flot concurrent maximal. Nous pouvons alors construire en 8.8 (b), le graphe étendu, qui modélise le problème sous forme de flot maximum. Il est important de noter que la capacité des arcs est elle-même dépendante du flot maximum $F$ (c'est une sorte de proportion à respecter).

L'extension que nous proposons à l'algorithme d'Edmonds-Karp comporte deux différences majeures :

(i) A la place d'un chemin augmentant, nous construisons ici une arborescence augmentante à chaque itération.
(ii) Le flot concurrent maximum que pourra supporter cette arborescence, dépendant des demandes spécifiques vers chaque terminal.

Nous utilisons à nouveau la notation $T(s)$ pour expliciter l'ensemble des destinations associée à la source unique $s$. Nous supposerons également que les capacités sont toutes strictement positives.

**Algorithm 16** : FCM mono-source

**Require:** un graphe orienté $G = (V, A)$ muni des capacités $(c_a)_{a \in A}$ et un ensemble de demandes $(s, t^k, d^k)_{k \in K}$.
**Ensure:** la valeur optimale $\gamma^*$ du flot concurrent maximal de $s$ vers $T(s)$.
1: Initialisation de $\gamma^* \leftarrow 0$.
2: **while** Il existe une arborescence de plus court chemin $\tau$ couvrant l'ensemble $T(s)$ dans le graphe d'écart **do**
3: Trouver le $\gamma$ maximum tel que le flot $\gamma d^k$ peut être acheminé sur chaque arc, sur la structure $\tau$ entre $s$ et chaque $t^k$ sans violer de contrainte de capacité.
4: Augmenter le(s) flot(s) selon $\gamma$, et mettre à jour le graphe
5: $\gamma^* \leftarrow \gamma^* + \gamma$
6: **end while**

A l'instar de l'algorithme d'Edmonds-Karp la valeur de $\gamma$ est augmentée itérativement par le calcul d'une arborescence de plus court chemin en nombre d'arcs. Ces arborescences sont calculées dans le *graphe d'écart* défini de la même manière que précédemment.

Le $\gamma$ maximum peut être obtenu facilement : il suffit de reporter les demandes sur chaque arc par lesquels cette demande sera routée. Cela génère sur chaque arc une demande cumulée donnée par la relation :

$$D_a = \sum_{k \in K} \sum_{\substack{p \in P^k \cap \tau \\ p \ni a}} d^k \tag{8.1}$$

Le $\gamma$ maximum deviendra alors simplement le minimum des ratios parmi tous les arcs (capacité résiduelle sur la demande cumulée- infinie si nulle) : soit $\gamma = \min_{a \in \tilde{A}} \frac{c_a}{D_a}$ . Au cours de l'étape 4, nous mettons à jour le graphe d'écart de la même manière que l'algorithme de flot maximum.

Nous illustrons l'algorithme sur l'exemple suivant. Nous nous munissons du graphe suivant 8.9. Nous calculons l'arborescence des plus courts chemins (en rouge sur la figure 8.10)avec la demande cumulée sur chaque arc. Ici nous avons $\gamma = \frac{2}{3}$. Nous mettons alors à jour le graphe d'écart (à droite de la figure 8.10). Nous répétons la recherche d'arborescence des plus courts chemins (en rouge sur la figure 8.11), avec la demande cumulée sur chaque arc. Nous avons ici $\gamma = 1$. Nous mettons alors à jour le graphe d'écart (à droite de la figure 8.11). Nous ne pouvons plus trouver d'arborescence augmentante, car le nœud 5 n'est plus atteignable. La solution optimale est de $\gamma^* = \frac{5}{3}$, égale à la densité de la coupe de densité minimale représentée en rouge.

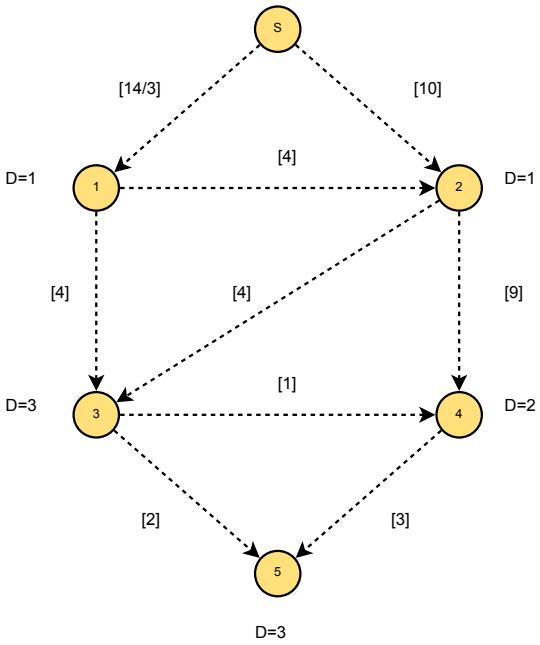



FIGURE 8.9 – *Graphe initial.*

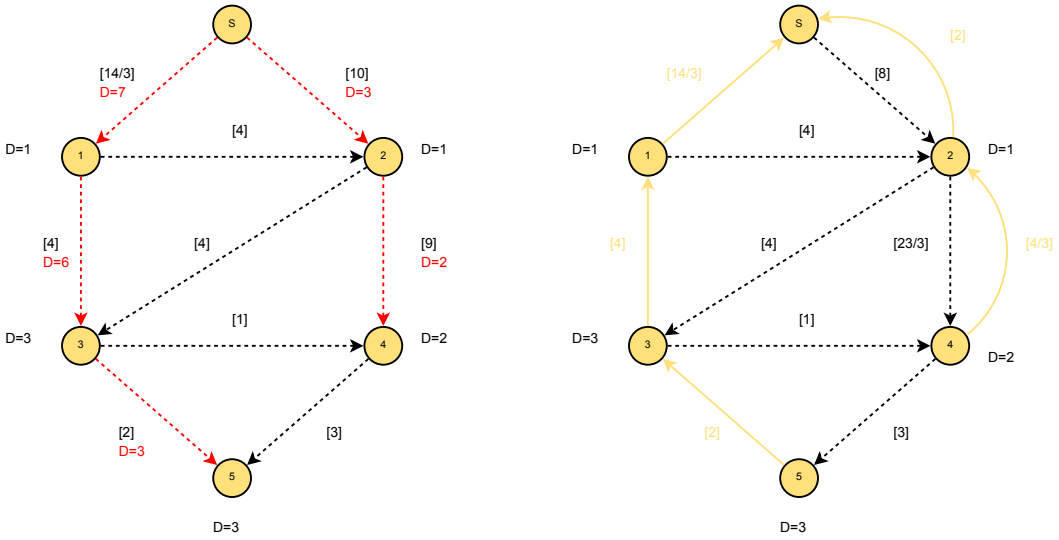



FIGURE 8.10 – *Itération 1.*

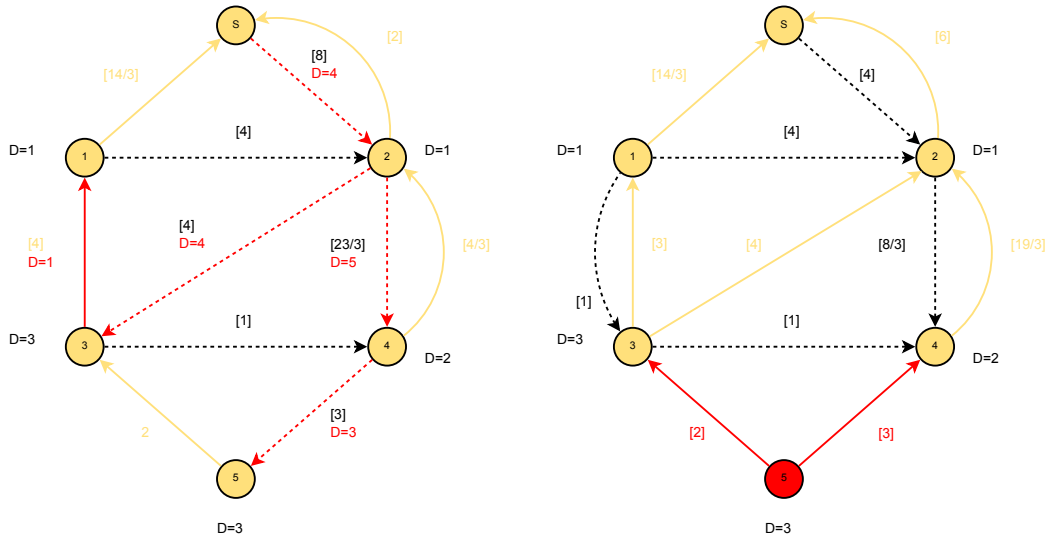



FIGURE 8.11 – *Itération 2.*

## 8.3 Preuve d'exactitude et de forte polynomialité

Avant de prouver l'exactitude de notre algorithme, nous allons rappeler quelques notations. Une coupe orientée dans le graphe $G$ peut être définie par un ensemble de nœuds $U \subset V$, ou bien de manière équivalente, par un ensemble d'arcs $\delta^+(U)$ divergents de $U$. Nous utiliserons alors parfois $\bar{U} = V \setminus U$ le complément de $U$.

Nous utiliserons également $c(U,\bar{U}) = \sum_{a\in\delta^+(U)} c_a$ et $d(U,\bar{U}) = \sum_{\substack{k\in K \\ s^k\in S, t^k\in\bar{U}}} d^k$. De manière analogue, lorsque nous manipulons des variables de flots $f_a$, $f(U,\bar{U})$ décrira la somme des flots sur les arcs de $\delta^+(U)$ : $f(U,\bar{U}) = \sum_{a\in\delta^+(U)} f_a$.

La densité d'une coupe $\delta^+(U)$ est définie comme le ratio des capacités sur la demande qu'elle déconnecte :

$$\chi(U,\bar{U}) = \frac{c(U,\bar{U})}{d(U,\bar{U})}. \tag{8.2}$$

Dans le cas général, il existe une propriété de dualité faible : toute valeur de solution de FCM $\gamma$ est inférieure à toute coupe de densité $\chi(U,\bar{U})$. C'est donc naturellement vrai pour la valeur optimale du FCM et la densité de la coupe de densité minimale :

$$\gamma^* \leq \chi^* = \min_{U\subset V}\{\chi(U,\bar{U}) :\ d(U,\bar{U}) \neq 0\}. \tag{8.3}$$

La dualité forte, elle, n'est pas vérifiée dans le cas général. De plus, le problème de déterminer une coupe de densité minimale (ou "Sparsest cut") est dans le cas général un problème NP-difficile, que la graphe soit orienté ou non. Un exemple où cette différence existe bien a été donné par Okamura-Seymour [81] (voir figure 8.12) dans un graphe non orienté ; il est généralement repris pour illustrer ce phénomène. De nombreuses études ont été menées afin de créer des algorithmes d'approximation pour la coupe de densitée minimum [48, 85, 69, 102], qui produisent des bornes très proches de l'optimum (par exemple $O(\sqrt{\log n})$ dans [5]).

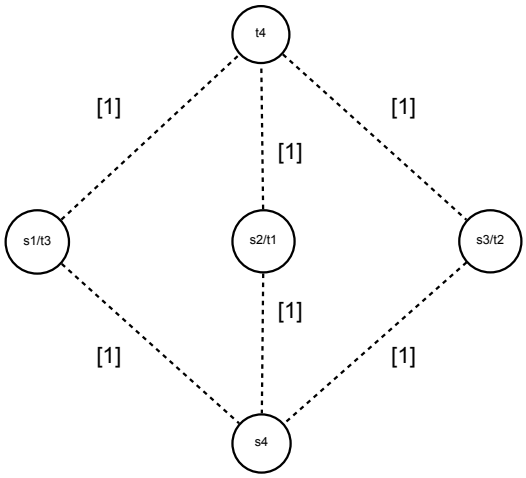



FIGURE 8.12 – *Exemple de Okamura et Seymour. Les commodités sont exprimées par leurs couples $s_k, t_k$ et ont pour valeur de demande 1. Le flot concurrent maximal est de $\frac{3}{4}$. La coupe de densité minimale est de* 1.

[67, 95] nous apprennent plus précisément que si le graphe de demandes (le graphe qui relie les nœuds $\{s^k, t^k\}$) ne contient pas de sous-graphe constitué de trois arêtes disjointes, ou un triangle et une arête disjointe, alors l'écart de dualité sera nul. Ce résultat généralise l'observation de [53] sur le problème de FCM muni de deux demandes. D'autres investigations ont été menées par [95] sur des cas particuliers où cet écart n'existe pas.

C'est le cas de notre problème FCM muni d'une seule source, où, utilisant le théorème de Flot-Max/Coupe-Min nous pouvons retrouver ce résultat facilement. Cependant, un résultat plus général établit que cet écart est nul lorsque deux nœuds $s$ et $t$ sont tels que pour chaque demande nous avons, $s^k = s$ ou $t^k = t$ [95]. Si cet écart est nul, il n'existe à notre connaissance, aucun algorithme fortement polynomial pour traiter ce cas.

Nous allons dès à présent prouver la forte polynomialité de notre algorithme :

**Théorème 25.** *Le flot concurrent maximal mono-source peut être résolu de façon optimale par l'algorithme 16 avec une complexité en $O(m^2 n)$.*

La preuve du théorème 25 est relativement proche de celle utilisée pour l'algorithme d'Edmonds-Karp.

**Lemme 3.** *Si l'algorithme 16 stoppe, alors avons obtenu $\gamma^*$ la solution optimale du flot concurrent maximal.*

*Démonstration.* : l'algorithme s'arrête lorsqu'il n'existe plus de façon de créer une arborescence couvrante $T(s)$, du fait des capacités, dans le graphe d'écart. Cela signifie qu'il existe un sous-ensemble de nœuds $\bar{U} \subset V \setminus \{s\}$, incluant une ou plusieurs destinations ($\bar{U} \cap T \neq \emptyset$) qui ne peuvent être atteintes depuis $s$. Plus précisément, il existe un ensemble $U$ tel que $f_a = c_a$ pour tout $a \in \delta^+(U)$ et $f_a = 0$ pour tout $a \in \delta^-(U)$ (le contraire signifierait qu'un arc de retour de capacité non nulle existe, et que de ce fait un des nœuds parmi $U$ peut être atteint). Nous déduisons alors que pour toutes les destinations $T(s) \cap \bar{U}$, tous les flots entrant sont utilisés pour satisfaire $\gamma^*$ fois la demande totale :

$f(U, \bar{U}) = \gamma^* d(U, \bar{U})$.

De plus nous avons $f(U, \bar{U}) = c(U, \bar{U})$. Il s'ensuit que $\gamma^* d(U, \bar{U}) = c(U, \bar{U})$ et $\gamma^* = \chi^*$ et puisque $\chi^*$ est une borne supérieure, $\gamma^*$ est donc optimal. □

Nous pouvons remarquer que s'il existe une arborescence couvrant $T(s)$, alors il existe également une arborescence couvrant tous les nœuds $V$ (les demandes "nulles" peuvent être acheminées sur des arcs de capacité nulle). Si ce n'était pas le cas, nous pourrions trouver un ensemble $U$ contenant $s$ et $T(s)$ tel que $\bar{U} \neq \emptyset$, $f(U, \bar{U}) = c(U, \bar{U})$ et $f(\bar{U}, U) = 0$. Ce qui est bien évidemment impossible puisque $\bar{U}$ ne contient pas de destinations. Cela signifie que pour toute itération qui n'est pas la dernière, il existe un chemin de $s$ vers tout nœud $v$. Nous appellerons $l_i(s, v)$ la longueur du plus court chemin (avec des poids unitaires) de $s$ vers $v$ à la $i$ème itération. Nous ferons également correspondre $G_i$ le graphe d'écart à la $i$ème itération (avant l'augmentation de flot).

**Lemme 4.** *La fonction $l_i(s, v)$ ne diminue pour aucun nœud $v \in V$, pour $i$ croissant.*

*Démonstration.* : Supposons la proposition fausse. Parmi tous les nœuds tels que

$$l_{i+1}(s, v) < l_i(s, v), \tag{8.4}$$

soit $v$ le nœud tel que $l_{i+1}(s, v)$ est le plus faible.

Soit $u$ le prédécesseur de $v$ sur le plus court chemin de $s$ vers $v$ dans $G_{i+1}$. Nous avons de façon évidente :

$$l_{i+1}(s, u) = l_{i+1}(s, v) - 1. \tag{8.5}$$

Puisque $l_{i+1}(s, u) < l_{i+1}(s, v)$, le plus court chemin de $s$ vers $u$ ne diminue pas de l'itération $i$ à l'itération $i + 1$ (par le choix de $v$). En d'autres termes nous avons :

$$l_i(s, u) \leq l_{i+1}(s, u). \tag{8.6}$$

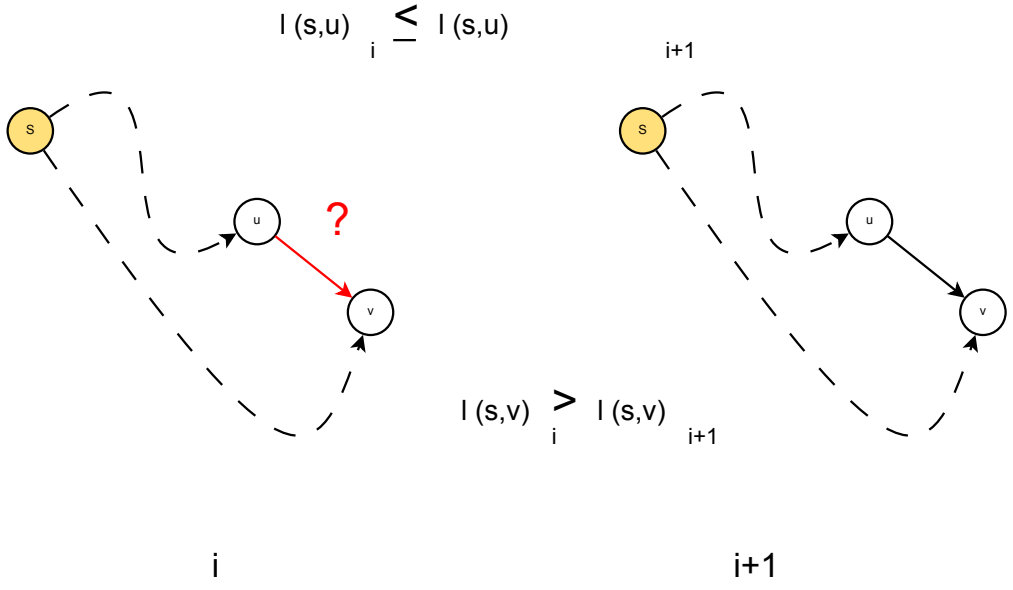



FIGURE 8.13 – $(u, v)$ *appartient-il à* $G_i$ *?*

Montrons que $(u, v)$ n'appartient pas à $G_i$. Pour ce faire, supposons le contraire : soit $(u, v)$ appartient à $G_i$. Nous pouvons alors écrire que $l_i(s, v) \leq l_i(s, u) + 1$. En utilisant (8.6), la précédente inégalité nous amène à $l_i(s, v) \leq 1 + l_{i+1}(s, u)$. Utilisant à présent (8.5), nous obtenons $l_i(s, v) \leq l_{i+1}(s, v)$ ce qui contredit (8.4).

Cela signifie donc que l'arc $(u, v)$ n'est pas présent dans $G_i$, mais apparaît donc seulement après l'augmentation de flot. Cela est possible uniquement si l'arc $(v, u)$ a été utilisé à l'itération $i$. En d'autre termes, $(v, u)$ appartient à l'arborescence de plus court chemin dans $G_i$. Cela signifie que $l_i(s, v) = l_i(s, u) - 1$. En utilisant (8.6), l'inégalité précédente nous donne $l_i(s, v) \leq l_{i+1}(s, u) - 1$. Sachant que (8.5), nous obtenons $l_i(s, v) \leq l_{i+1}(s, v) - 2$ ce qui contredit (8.4). □

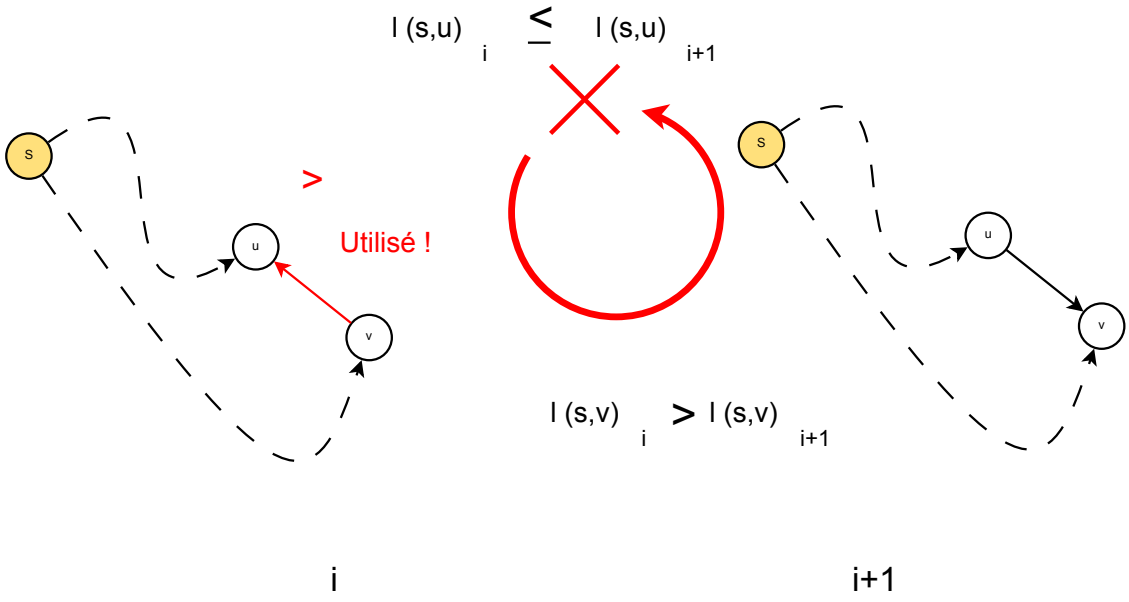



FIGURE 8.14 – *Contradiction.*

Un arc critique $a$ dont la capacité est le facteur limitant pour l'arborescence de plus court chemin $\tau$ (i.e., un arc minimisant le ratio de la capacité sur la demande cumulée $\sum_{k \in K} \sum_{\substack{p \in P^k \cap \tau \\ p \ni a}} d^k$). L'arborescence contient au minimum un arc critique. Il est alors évident qu'un arc critique à l'itération $i$, ne sera plus présent à l'itération $i + 1$.

Nous allons à présent exhiber la propriété suivante :

**Lemme 5.** *Un arc ne peut être critique plus de* $\frac{n}{2}$ *fois.*

*Démonstration.* : Supposons $(u, v)$ critique à l'itération $i$. Il est important de rappeler que $(u, v)$ peut être alors un arc de $A$ ou un arc de retour associé à un arc de $A$. Puisque $(u, v)$ appartient à l'arborescence des plus courts chemins, nous pouvons écrire que

$$l_i(s, v) = l_i(s, u) + 1. \tag{8.7}$$

Supposons que $(u, v)$ est également critique à l'itération $j > i$. Cela signifie que $(v, u)$ doit être utilisé au moins une fois entre l'itération $i$ et $j$. En d'autres termes, il existe $k$ $(i < k < j)$

tel que

$$l_k(s,u) = l_k(s,v) + 1. \tag{8.8}$$

Or nous savons du lemme précédent que $l_k(s,v) \geq l_i(s,v)$. De cette inégalité et de (8.7) et (8.8), nous déduisons

$$l_k(s,u) \geq l_i(s,u) + 2. \tag{8.9}$$

En d'autres mots, après que $(u,v)$ eut été critique lors d'une itération ,la distance la plus courte entre $s$ et $u$ augmente d'au moins de 2. Puisque la distance maximale entre ces deux nœuds ne peut excéder $n-1$ (sauf à la dernière itération où cette distance peut être considérée infinie), $(u,v)$ ne peut pas être critique plus de $\frac{n}{2}$ fois. □

La preuve du théorème 25 découle naturellement. Puisqu'un arc ne peut être critique plus de $\frac{n}{2}$ itérations et que le nombre d'arcs ne peut excéder $2m$ (en comptant également les arcs de retour), l'algorithme 16 doit nécessairement s'arrêter au bout de $O(mn)$ itérations au maximum. De plus, la génération d'arborescence peut être effectuée avec une complexité de $O(m)$ par un algorithme de recherche "breadth-first".

Il est important de noter que 16 est également capable de déterminer la coupe de densité minimale dans ce dernier cas. Par exemple, si $T(s) = V \setminus \{s\}$ et $d^k = 1$ pour $k \in K$, alors nous pouvons déterminer $\delta^+(U)$ tel que $s \in U$ minimise $\frac{c(U,\overline{U})}{|\overline{U}|}$.

**Corollaire 1.** *L'algorithme 16 détermine une coupe de densité minimale en une complexité de $O(m^2 n)$ (dans le cas mono-source).*

**Remarque 5.** *Même si nous avons supposé un graphe orienté, l'algorithme 16 peut également s'appliquer de la même manière à un graphe non orienté, sans nécessairement passer par la transformation explicitée précédemment. De manière analogue au problème de flot maximum, il est possible de remplacer chaque arête $\{u,v\}$ de capacité $c$ par deux arcs $(u,v)$ et $(v,u)$ munis chacun d'une capacité $c$. Puisque dans le cas mono-source nous pouvons toujours annuler du flot pour nous assurer qu'au moins un des deux arcs $(u,v)$ et $(v,u)$ ne contient aucun flot. De ce fait remplacer une arête par deux arcs ne change pas la valeur du flot concurrent maximal.*

## 8.4 Expérimentations

Le tableau 8.2 est dédié au FCM dans sa version mono-source (instance 8). L'instance résolue par notre algorithme combinatoire ne peut pas être résolue par les formulations précédentes ($FCM_{Chemin}$, $FCM_{Arbre}$, $FCM_{Agg}$ et $FCM_{Gener}$).

| CPU time (s) | Itérations |
|---|---|
| 107987 | 95341 |

TABLE 8.2 – La solution du FCM résolu avec l'algorithme combinatoire (instance 8 avec $|V| = 125000$, $|S| = 1$, $|K| = 124999$, and $|A| = 735000$)

Le tableau 8.3 reporte une comparaison entre la formulation arborescente et l'algorithme fortement polynomial. On remarque que plus l'instance est élevée, plus l'écart est élevé. On voit très nettement la performance de l'algorithme polynomial qui est rapidement 1000 fois plus rapide que la formulation. Il est intéressant de noter que le nombre d'itérations reste assez comparable, mais finit par être plus faible pour le programme linéaire.

| a=b | \|V\| | CPU time (s) | | Iterations | |
|---|---|---|---|---|---|
| | | Algorithme Polynomial | Arbre | Algorithme Polynomial | Arbre |
| 7 | 343 | 0,01 | 36 | 176 | 197 |
| 8 | 512 | 0,1 | 140 | 262 | 294 |
| 9 | 729 | 0,3 | 318 | 432 | 349 |
| 10 | 1000 | 0,6 | 1673 | 656 | 525 |
| 11 | 1331 | 1,2 | 6601 | 782 | 679 |

TABLE 8.3 – Comparaison de résolution du problème de flot concurrent maximal dans le cas mono-source sur RMF. Tous les nœuds sont destinations.

# Conclusion

Nous avons étudié le problème de flot concurrent maximal dans un cas particulier où il n'y a qu'une seule source (mais potentiellement plusieurs destinations). Pour le résoudre nous avons alors présenté un algorithme fortement polynomial doté d'une complexité de $O(|A|^2.|V|)$, la meilleure constatée à notre connaissance pour ce cas particulier. Nous pouvons nous apercevoir que nous avons généralisé l'algorithme d'Edmonds-Karp, avec une complexité similaire. Les performances pratiques sont, sans grande surprise, bien plus importantes que les meilleures générations de colonnes présentées précédemment. On pourrait alors se demander s'il n'existe pas une manière de renforcer la complexité de cet algorithme en s'inspirant des autres travaux sur le problème de flot maximal. Il n'est cependant pas trivial de les utiliser tels qu'ils sont, puisque la plupart s'appuie sur des "pré-flots", qui ne garantissent pas l'équité de la satisfaction de chacune des demandes *à priori*. Un autre aspect qui demeure toujours en cours d'étude, est de généraliser cet algorithme pour le cas multi-sources et multi-destinations.

# Conclusion et perspectives

Nous nous sommes placés dans une approche d'un constat immédiat, où nous voulons maximiser le débit/qualité de service client sur un réseau existant, par un routage optimal : nous essayons de tirer parti au mieux des capacités de notre réseau. Ce problème, le flot concurrent maximal, est un problème bien connu dans la littérature. Même si ce problème possède une complexité polynomiale, il est en pratique difficile à résoudre sur de grandes instances. Nous avons donc proposé une manière particulière, qui s'étend au-delà des arborescences, pour pouvoir manipuler ces grands graphes.

Nous avons également exhibé un algorithme fortement polynomial pour résoudre le problème de flot concurrent maximal dans sa version mono-source, dont les performances sont radicalement supérieures à tous les modèles linéaires explicités ici. Il permet de résoudre également plusieurs problèmes connexes (problèmes d'affectations notamment). Sa complexité est à notre connaissance la meilleure qui existe dans la littérature pour ce cas particulier.

Une autre perspective qui découle naturellement de ce cheminement est de trouver une extension à cet algorithme pour traiter le cas général. Ce cas général ne garantissant plus certaines propriétés spécifiques (comme l'absence de saut de dualité), cette extension ne semble pas triviale.

# Conclusion générale

Nous avons traité dans la première partie des problématiques de localisation de caches dans des réseaux de distribution de contenus, selon des considérations économiques et des scénarios réalistes. Nous avons présenté différentes méthodes de programmation dynamique existantes pour des types de problèmes de localisation dans les arborescences. Nous avons ensuite proposé une nouvelle approche de programmation dynamique et un modèle linéaire pour répondre au problème de localisation de caches transparents dans un réseau arborescent. Nous avons également montré qu'il était possible de produire des algorithmes polynomiaux exacts pour différentes version du problème de déploiement de réseau (notamment $k$-médian) moyennant des hypothèses simplificatrices.

Cette étude s'articule alors en deux points complémentaires ; elle permet d'abord de donner des indications sur la pertinence de ces méthodes sur des problèmes particuliers. Le paradigme de programmation dynamique que nous proposons, portant sur des ensembles de solutions, semble particulièrement approprié sur des structures arborescentes et des problèmes aux contraintes complexes. Même si a priori ces algorithmes ont des complexités théoriquement bornées par des nombres exponentiels, l'expérimentation pratique est satisfaisante. De futures études permettraient dès lors de mieux comprendre analytiquement pour quelles raisons ces performances restent intéressantes, et si d'autres structures d'application (comme un graphe quelconque) pourraient être envisagées.

Le second point concerne de façon plus générale, les leviers d'aide à la décision sur des problématiques de déploiements, selon des hypothèses différentes. Les algorithmes ainsi développés, permettent de répondre rapidement à des questions logistiques selon des axes stratégiques et/ou d'investissement envisagés. De plus amples études devraient être apportées pour circonscrire au maximum les futurs possibles des réseaux de distribution de contenus.

La seconde partie s'oriente sur le constat immédiat des lacunes d'un réseau au niveau de ses capacités ; les routages optimaux et les liaisons critiques. C'est le problème de Flot Concurrent Maximal (FCM). En considérant qu'un cache permet de diminuer le trafic de ses liens en amont, la localisation de ces liens offrent implicitement un conseil de localisation, afin de pallier cette faiblesse constatée. Nous proposons alors des formulations génériques pour modéliser le FCM, applicables à bien d'autres problèmes connexes, et nous démontrons leurs intérêts pratiques sur le problème de flot concurrent maximal. Sur ce dernier problème, nous apportons un algorithme dédié au cas mono-source, dont les performances aussi bien théoriques que pratiques dépassent les meilleurs modèles de la littérature —ceux présentés inclus. Nous pouvons alors nous demander d'abord s'il existe des moyens de construire des formulations encore plus performantes adaptées à des problèmes de multi-flots spécifiques. De nouvelles études pourraient permettre de construire des algorithmes fortement polynomiaux avec des complexités encore meilleures pour le cas du flot concurrent maximal muni d'une seule source. Enfin, s'il est possible d'étendre cet algorithme pour résoudre le problème Flot Concurrent Maximal dans sa version générale.

L'horizon temporel est ce qui sépare ces deux problèmes complémentaires. Dans le premier cas, on s'intéresse à un investissement sur le long terme selon une estimation plus ou moins précise de la demande ; il s'agit donc de la satisfaire à moindre coût. Les capacités représentant uniquement des contraintes sur les investissements minimums.

Dans le second cas, on mesure plutôt la capacité effective de notre réseau, par des tables de routage partielles optimales. Ceci nous permet de connaître à quel moment notre réseau n'est plus en mesure de satisfaire ses usages. Du point de vue de l'investissement, on pourrait alors voir cela

comme un risque sur le volume global supplémentaire qui pourrait survenir, soit sur un horizon de temps plus important, soit parce que les données des demandes sont incertaines. On peut voir alors ce problème comme un problème de recours. Il pourrait être intéressant d'étudier ces problèmes de manière conjointe. De plus nous pourrions caractériser l'incertitude de la demande de manière plus fine comme le font les auteurs [103], ce qui reviendrait à caractériser le problème sous sa forme robuste.

# Publications

## Congrès Nationaux

[107] P-O. Bauguion, W. Ben Ameur, E. Gourdin. Formulation arborescente performante pour les problèmes de multi-flots. *ROADEF 2012*, 2012.

[108] P-O. Bauguion, W. Ben Ameur, E. Gourdin. Localisation de caches dans le réseau de distribution de contenus. *ROADEF 2013*, 2013.

## Congrès Internationaux

[109] P-O. Bauguion, W. Ben Ameur, E. Gourdin. Cache location in tree networks (preliminary results). *INOC 2011-Lecture Notes in Computer Sciences*, 5 :517-522, 2011.

[110] P-O. Bauguion, W. Ben Ameur, E. Gourdin. Cache location in content delivery networks. *EWGLA*, 2011.

[111] P-O. Bauguion, W. Ben Ameur, E. Gourdin. Efficient tree-based model for multicommodity flow problems. *ISCO 2012 - proceedings*, 2012.

[112] P-O. Bauguion, W. Ben Ameur, E. Gourdin. New formulations for multicommodity flow problems and strongly polynomial algorithm for Maximum Concurrent Flow Single Source. *INOC 2013 - proceedings*, 2013.

## Journal

[113] P-O. Bauguion, W. Ben Ameur, E. Gourdin. Efficient algorithms for the maximum concurrent flow problem. *Networks*,-.

# Bibliographie


[1] Sndlib : Survivable network design library.

[2] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4) :1204 –1216, jul 2000.

[3] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows : theory, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[4] Robert E. Tarjan Andrew V. Goldberg. A new approach to the maximum flow problem. In *Annual ACM Symposium on Theory of Computing*, pages 136—-146, 1986.

[5] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *ACM STOC'04*, 2004.

[6] Dan Asit and Don Towsley. An approximate analysis of the LRU and FIFO ; buffer replacement schemes. *SIGMETRICS Perform. Eval. Rev.*, 18(1) :143–152, 1990.

[7] Y. Auman and Y. Rabani. Approximate min-cut max-flow theorem and approximations algorithm. *SIAM Journal on Computing*, 27 :213–238, 1998.

[8] S.A. Plotkin A.V. Goldberg, J.D. Oldham and C. Stein. An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity. *IPCO*, pages 338–352, 1988.

[9] Pasquale Avella, Maurizio Boccia, Roberto Canonico, Donato Emma, Antonio Sforza, and Giorgio Ventre. Web cache location and network design in VPNs, 2003.

[10] F. Barahona and A.R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36(2) :157–173, 1986.

[11] L.A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5 :78–101, 1966.

[12] W. Ben-Ameur and J. Neto. Acceleration of cutting-plane and column generation algorithms : Applications to network design. *Networks*, 49(1) :3–17, 2007.

[13] Farid Benbadis, Nidhi Hegde, Fabien Mathieu, and Diego Perino. Playing with the bandwidth conservation law. In *Proceedings of the 8th International Conference on Peer-to-Peer Computing*, 2008.

[14] D. Bienstock. Potential function methods for approximatively solving linear programs : theory and practice. Technical report, CORE Lecture Series Monograph, 2001.

[15] D. Bienstock and O. Raskina. Asymptotic analysis of the flow deviation method for the maximum concurrent flow problem. *Mathematical Programming, Ser. B*, 91 :479–492, 2002.

[16] Yacine Boufkhad, Fabien Mathieu, Fabien de Montgolfier, Diego Perino, and Laurent Viennot. Achievable catalog size in peer-to-peer video-on-demand systems. In *Proceedings of the Seventh International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[17] S. Khanna C. Chekuri and F.B. Shepherd. The all-or-nothing multicommodity flow problem. In *STOC'04*, 2004.

[18] J. Roberts C. Fricker, P. Robert and N. Sbihi. Impact of traffic mix on caching performance in a content-centric network. *CoRR*, 1202.0108, 2012.

[19] S.A. Plotkin C.H. Norton and E. Tardos. Using separation algorithms in fixed dimension. *J. Algorithms*, 13 :79–98, 1992.

[20] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *In Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 378–388, 1999.

[21] C. Chekuri and S. Khanna. Edge disjoint paths revisited. In *SODA'03*, pages 628–637, 2003.

[22] M. Chrobak, L.L. Larmore, and W. Rytter. The $k$-median problem for directed trees. *Lecture Notes in Computer Science*, 2136 :260–271, 2001.

[23] E. Cronin, S. Jamin, C. Danny, and R. Yuval. Constrained mirror placement on the Internet, 2002.

[24] O. Günlük D. Bienstock, S. Chopra and C.-Y. Tsai. Minimum cost capacity installation for multicommodity network flows. *Mathematical Programming*, 81 :177–199, 1998.

[25] G. B. Dantzig and Wolfe. Decomposition principle for linear programs. 8 :101–111, 1960.

[26] P.B. Dantzig, R.S. Hall, and M.F. Schwartz. A case for caching file objects inside Internetworks. *SIGCOMM '93*, pages 239–243, 1993.

[27] U. Derigs and W. Meier. Implementing Goldberg's max-flow-algorithm — a computational investigation. *ZOR — Methods and Models of Operations Research*, 33 :383–403, 1989.

[28] M. Deza and M. Laurent. *Geometry of cuts and metrics*, volume 15. Springer, 1997.

[29] Yefim Dinitz. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Doklady Akademii nauk SSSR*, 11 :1277–1280, 1970.

[30] C.H. Papadimitriou P.D. Seymour E. Dahlhaus, D.S. Johnson and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23 :864–894, 1994.

[31] J. Edmonds. Submodular functions, matroids, and certain polyhedra. In *Calgary International Conference on Combinatorial Structures and Their Applications*, pages 69–87, 1970.

[32] J. Edmonds. Edge-disjoint branchings. In B. Rustin, editor, *Combinatorial Algorithms*, pages 91–96. Academic Press, 1973.

[33] J. Edmonds. Combinatorial optimization - eureka, you shrink ! In G. Rinaldi M. Juenger, G. Reinelt, editor, *Lecture Notes in Computer Science 2570*, pages 11–26. Springer, 2003.

[34] J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problem. *Journal of the ACM*, 19(2) :248–264, 1971.

[35] D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, page 992–1009, 1978.

[36] M. Franck et P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3, 1956.

[37] Didier Fayard and Gerard Plateau. An exact algorithm for the 0-1 collapsing knapsack problem. *Discrete Applied Mathematics*, 49 :175–187, 1994.

[38] FICO, Dash Optimization. *Xpress-MP Optimizer Reference Manual*, 2008.

[39] Philippe Flajolet, Danièle Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Appl. Math.*, 39(3) :207–229, 1992.

[40] L. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4) :505–520, 2000.

[41] András Frank. A weighted matroid intersection algorithm. *Journal of Algorithms*, 2(4) :328–336, 1981.

[42] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *FOCS*, pages 300–309, 1998.

[43] Erol Gelenbe. A unified approach to the evaluation of a class of replacement algorithms. *IEEE Trans. Comput.*, 22(6) :611–618, 1973.

[44] Boris Goldengorin, Diptesh Ghosh, and Gerard Sierksma. Branch and peg algorithms for the simple plant location problem. *Computers & Operations Research*, 30 :967–981, 2003.

[45] E. Gourdin, M. Labbé, and H. Yaman. Telecommunication and location. In Z. Drezner and H.W. Hamacher, editors, *Facility Location : Applications and Theory*, pages 275–305. Springer, 2002.

[46] Alexander Schrijver Grötschel Martin, Laszlo Lovasz. Complexity, oracles, and numerical computation. In *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

[47] S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, 1 :603–612, 2000.

[48] O. Günlük. On the min-cut max-flow ratio for multicommodity flows. *SIAM J. of Disc. Math.*, 21 :1–15, 2007.

[49] M.T. Hajiaghayi and T. Leighton. On the max-flow min-cut ratio for directed multicommodity flows. *Theor. Comput. Sci.*, 352, 2006.

[50] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12 :450–459, 1964.

[51] S. L. Hakimi. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13 :462–475, 1965.

[52] S.L. Hakimi and E.F. Schmeichel. Locating replicas of a database on a network. *Networks*, 30(1) :31–36, 1997.

[53] T.C. Hu. Multi-commodity network flows. *Operations Research*, 11 :344–360, 1963.

[54] C. Huang, J. Li, and K.W. Ross. Can Internet video-on-demand be profitable ? In *SIGCOMM '07 : Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 133–144, New York, NY, USA, 2007. ACM.

[55] Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. On the placement of Internet instrumentation. In *INFOCOM (1)*, pages 295–304, 2000.

[56] P. R. Jelenkovic. Asymptotic approximation of the move-to-front search cost distribution and least-recently-used caching fault probabilities. *Annals of Applied Probability*, 9(2) :430–464, 1999.

[57] K.L. Jones, I.J. Lustig, J.M. Farvolden, and W.B. Powell. Multicommodity network flows : The impact of formulation on decomposition. *Mathematical Programming*, 62(1-3) :95–117.

[58] A. Jüttner. Optimization with additional variables and constraints. *Operations Research Letters*, 33 :305–011, 2005.

[59] G. Karakostas. Faster approximation schemes for fractional multicommodity flow problems. *ACM Transactions on Algorithms*, 4(1), 2008.

[60] O. Kariv and L. Hakimi. An algorithmic approach to network location problems. ii : The $p$-median. *SIAM J. Appl. Math.*, 37(3) :539–560, 1979.

[61] P. Krishnan, Danny Raz, and Yuval Shavitt. The cache location problem. *IEEE/ACM Transactions on Networking*, 8(5) :568–582, 2000.

[62] Joseph Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7 :48–50, 1956.

[63] Eugene L. Lawler. Matroid intersection algorithms. *Mathematical Programming*, 9 :31–56, 1975.

[64] Nevena Lazic, Brendan J. Frey, and Parham Aarabi. Solving the uncapacitated facility location problem using message passing algorithms.

[65] Youngho Lee, Seong in Kim, Seungjun Lee, and Kugchang Kang. A location-routing problem in designing optical Internet access with WDM systems. *Photonic Network Communications*, 6(2) :151–160, 2003.

[66] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *29th Annual IEEE Symposium on Foundations of Computer Science*, pages 215–245, 1998.

[67] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *JACM*, 46 :215–245, 1999.

[68] Bo Li, M.J. Golin, G.F. Italiano, Xin Deng, and K. Sohraby. On the optimal placement of web proxies in the Internet. *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 3 :1282–1290 vol.3, 1999.

[69] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15 :215–245, 1995.

[70] M.E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53 :1007–1023, 2005.

[71] Mario Gerla et Leonard Kleinrock Luigi Fratta. The flow deviation method : An approach to store-and-forward communication network design. 3 :97–133, 1973.

[72] H. Luss. Optimal content distribution in video-on-demand tree networks. *IEEE Transactions on systems, man, and cybernetics - part A : systems and humans*, 40(1), 2010).

[73] C. Hervet M. Chardy and D. Bossia. A dynamic programming approach for passive optical network design in tree graphs. In *Lecture Notes in Management Science*, volume 5, pages 93–100, 2013.

[74] S.C. Cosmadakis M. Yannakakis, P.C. Kanellakis and C.H. Papadimitriou. Cutting and partitioning a graph after a fixed pattern. In *10th Intl. Coll. on Automata, Languages and Programming*, pages 712–722, 1983.

[75] P. Mahey. Decomposition methods for mathematical programming. In Pardalos and Resende, editors, *Handbook of Applied Optimization*, pages 11–26. Oxford U. Press, 2002.

[76] D.W. Matula and F. Shahrokhi. The maximum concurrent flow problem and sparsest cuts. Technical report, Southern Methodist Univ. Dallas, 1986.

[77] D.W. Matula and F. Shahrokhi. The maximum concurrent flow problem. *Association for Computing Machinery*, 37, 1990.

[78] D.W. Matula and F. Shahrokhi. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Maths.*, 27 :113–123, 1990.

[79] Sanjay Melkote and Mark S. Daskin. Capacitated facility location/network design problems. *European Journal of Operational Research*, 129 :481–495, 2001.

[80] Sanjay Melkote and Mark S. Daskin. An integrated model of facility location and transportation network design. *Transportation Research Part A : Policy and Practice*, 35(6) :515–538, 2001.

[81] H. Okamura and P.D. Seymour. Multicommodity flows in planar graphs. *J. Combin. Theory Ser. B*, 31 :75–81, 1981.

[82] Nathalie Omnes. Intégration de technologies peer-to-peer pour les services audiovisuels. Master's thesis, ENST / FT R&D, 2007.

[83] R. Ravi S. Rao P. Klein, A. Agrawal. Approximation through multicommodity flow. In *31st Annual IEEE Symposium on Foundations of Computer Science*, pages 726–737, 1990.

[84] A. M. K. Pathan and R. Buyya. A taxonomy and survey of CDNs. Technical report, The University of Melbourne, 2007.

[85] S. Plotkin and E. Tardos. Improved bounds on the max-flow min-cut ratio for multicommodity flows. In *25th Symposium on Theory of Computing*, 1993.

[86] Stefan Podlipnig and Laszlo Böszömenyi. A survey of Web cache replacement strategies. *ACM Computing Surveys (CSUR)*, 35(4) :374–398, 2003.

[87] L. Qiu, V.N. Padmanabhan, and G.M. Voelker. On the placement of web server replicas. In *INFOCOM*, pages 1587–1596, 2001.

[88] Wenyu Qu, Keqiu Li, Masaru Kitsuregawa, and Takashi Nanya. An optimal solution for caching multimedia objects in transcoding proxies. *Computer Communications*, 30(8) :1802–1810, 2007.

[89] P. Radoslavov, R. Govindan, and D. Estrin. Topology-informed Internet replica placement. In *Proceedings of WCW'01 : Web Caching and Content Distribution Workshop, Boston, MA*, June 2001.

[90] J. Reese. Solution methods for the $p$-median problem : An annotated bibliography. *Networks*, 19 :125–142, 2006.

[91] D.R. Slutz R.L. Mattson, J. Gecsei and I.L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9 :78–117, 1970.

[92] L.A. Wolsey R.L. Rardin. Valid inequalities and projecting the multicommodity extended formulation for uncapacitated fixed charge network flow problems. *European Journal of Operations Research*, nov 1993.

[93] N. Robertson and P.D. Seymour. Graph minors xiii. the disjoint paths problem. *Journal of Comb. Theory Series B*, 63 :65–110, 1995.

[94] A. Schrijver. Homotopic routing methods. In H.J. Prömel B. Korte, L. Lovász and A. Schrijver, editors, *Paths, Flows and VLSI-Layout*, pages 329–334. Springer, 1990.

[95] A. Schrijver. *Combinatorial Optimization. Polyhedra and Efficiency*, volume 24. Springer, 2003.

[96] F. Shahrokhi and D.W. Matula. On solving large maximum concurrent flow problems. *Journal of the ACM*, 37 :318–334, 1991.

[97] D.B. Shmoys. Cut problems and their application to divide-and-conquer. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 192–235. PWS Publishing Company, 1997.

[98] A. Tamir. An $\mathcal{O}(pn^2)$ algorithm for the $p$-median and other related problems on tree graphs. *Operations Research Letters*, 19 :59–64, 1996.

[99] X. Tang and J. Xu. On replica placement for QoS-aware content distribution, 2004.

[100] Xueyan Tang and Samuel T. Chanson. Coordinated en-route web caching. *IEEE Transactions on Computers*, 51(6) :595–607, 2002.

[101] Lucia M. A. Drummond Tiago Araujo Neves, Luiz Satoru Ochi and Eduardo Uchoa e Celio Albuquerque. Optimization in content distribution networks. *EngOpt*, 2008.

[102] S. Tragoudas. Improved approximations for the min-cut max-flow ratio and the flux. *Mathematical Systems Theory*, 29 :157–167, 1996.

[103] H. Kerivin W. Ben-Ameur. Routing of uncertain traffic demands. *Optimization and Engineering*, 6 :283–313, 2005.

[104] Hassler Whitney. The abstract properties of linear dependence. *Am. J.Math.*, 57, 1935.

[105] P. Olivier Y. Carlinet, B. Kauffmann and A. Simonian. Impact of request correlations on the performance of multimedia caching. *Rapport interne Orange Labs.*, 2012.

[106] H. Yaman. *Concentrator Location in Telecommunications Networks*. Springer, 2005.