Résolution d'un problème de conception de réseau avec Parallel Solver

Raphaël Bernhard* — **Jacques Chambon*** — **Claude Lepape**** — **Laurent Perron**** — **Jean Charles Régin****

*France Télécom R&D**
905 rue Albert Einstein
F-06921 Sophia-Antipolis
{raphael.bernard, jacques.chambon}@francetelecom.com

*ILOG SA***
9 rue de Verdun
F-94253 Gentilly Cedex
{clepape,lperron,jcregin}@ilog.fr

RÉSUMÉ. *L'activité de conception d'un réseau pour un client donné nécessite la prise en compte de diverses contraintes dépendant du contexte et du client considéré. Dans cet article, nous rendons public un « benchmark » permettant d'évaluer des algorithmes sur de multiples variantes d'un problème de conception de réseau donné. Nos premières expériences montrent que des résultats satisfaisants peuvent être obtenus en intégrant des algorithmes de graphes au sein d'un outil de programmation par contraintes comme ILOG Solver. Par ailleurs, une meilleure répartition et la parallélisation de l'effort de recherche (grâce à ILOG Parallel Solver) permettent d'améliorer significativement ces résultats.*

ABSTRACT. *Designing a customized communications network requires addressing a wide variety of constraints that depend on the specific context and wishes of the intended client. In this article, we make public a benchmark for the evaluation of algorithms on multiple variations of a network design problem. Our initial experiments reveal that good results can be obtained through the integration of graph algorithms and a constraint programming engine like ILOG Solver. Significant improvements over these results can be obtained through a customized search procedure and the parallelization of the search using ILOG Parallel Solver.*

MOTS-CLÉS : *Conception de réseau, graphes, contraintes, parallélisme, procédure de recherche, génération de colonnes, programmation entière*

KEYWORDS: *Network Desing, Graphs, Constraints, Parallelism, Search Procedure, Column Generation, Integer Programming*

1re soumission à *JFPLC 2002*, le 18 avril 2002.

1. Introduction

Les moyennes et grandes entreprises échangent de plus en plus de données sur les réseaux connectant leurs sites. Par conséquent l'achat ou l'évolution d'un réseau est devenu un élément stratégique pour ces entreprises. Elles y consacrent une part toujours plus importante de leur budget de fonctionnement et ceci les obligent à être de plus en plus exigeantes vis à vis des opérateurs de réseau : lors de l'achat ou d'évolutions de leur réseau, elles font jouer fortement la concurrence et imposent aux opérateurs des contraintes qui ne peuvent pas toujours être anticipées par ce dernier.

Pour répondre à cette demande, France Télécom a développé une famille d'outils d'optimisation. A partir d'une demande de trafic à écouler entre un ensemble de sites, ces outils cherchent à minimiser la facture du client en jouant sur des paramètres tels que le débit et la longueur des liens, la manière de router le trafic, etc. Très généralement, à une offre commerciale correspond un outil d'optimisation ad-hoc, développé à ce jour au moyen de techniques de recherche opérationnelle. Cette démarche permet de disposer d'outils efficaces sur le plan du temps de calcul et de la qualité du résultat, mais elle est insuffisante sur deux points :

– Le temps de développement des outils ad-hoc est très largement supérieur au temps de réactivité du marché.

– Ces outils ne sont pas prévus pour prendre en compte des contraintes additionnelles non anticipées, que le client peut imposer dans son cahier des charges.

Dans le cadre du projet RNRT ROCOCO, nous étudions la possibilité de combiner l'approche « recherche opérationnelle » (RO) pour son efficacité en temps de calcul avec l'approche « programmation par contraintes » (PPC). L'objectif recherché vise à apporter une solution aux deux points soulevés ci-dessus :

– Un outil d'optimisation ne diffère d'un autre que par l'ensemble des contraintes qu'il prend en compte, et tous deux sont bâtis sur un noyau commun.

– Les contraintes spécifiques à chaque client sont prises en compte par simple adjonction de contraintes supplémentaires exprimées en PPC. Ces contraintes s'ajoutent naturellement au modèle initial, car leur formalisme d'expression est déclaratif et non-intrusif.

Un premier prototype de noyau commun a été élaboré. Ce prototype s'appuie sur des algorithmes de graphe intégrés à l'outil de programmation par contraintes ILOG Solver[SOL 01]. Ce prototype permet de décrire très aisément des contraintes supplémentaires spécifiant, par exemple, que tous les sites du réseau calculé doivent avoir exactement deux liens incidents et deux liens émergents. Ceci se traduit par la simple adjonction de quelques lignes dans le programme. L'effet de cette contrainte est de forcer l'outil à générer une topologie de boucle. Ainsi, cette simple manipulation logicielle permet de disposer, le temps d'une compilation, d'un outil couvrant l'offre commerciale concernant les boucles optiques haut débit.

Il est évident que ce type d'approche offre une très grande souplesse en terme de génie logiciel. Cependant, il faut impérativement éviter que cela ne se fasse trop aux dépens des performances et de la qualité des résultats. Afin de mesurer ces éléments, un « benchmark » a été mis au point représentant un ensemble de cas de réseau client dont il faut minimiser le coût sous différentes contraintes. Contrairement à la plupart des benchmarks utilisés en recherche opérationelle, l'objectif est ici non pas d'obtenir les meilleures performances possibles pour une version délimitée du problème, mais d'obtenir les meilleures performances en moyenne pour une multitude de variantes du problème, correspondant à la présence ou à l'absence de diverses contraintes additionnelles.

Les premiers résultats expérimentaux ont montré que notre prototype initial donnait, de ce point de vue, de bons résultats. L'analyse de ces résultats nous a néanmoins amené à rechercher diverses améliorations dans la manière de conduire la recherche. Deux types d'amélioration ont été considérés :

– l'utilisation de la version parallèle d'ILOG Solver, permettant d'utiliser la puissance des machines multi-processeurs ;

– la définition d'un réglage « fin » de la procédure de recherche, sur la base de diverses caractéristiques de l'arbre de recherche dévéloppé par le prototype initial.

L'objectif de cet article est de présenter les résultats qui nous paraissent susceptibles d'intéresser l'ensemble de la communauté RO et PPC. Ceci comprend :

– le benchmark, que nous avons décidé de rendre public (Section 2) ;

– le type de couplage que nous avons été amené à réaliser entre programmation par contraintes et algorithmes de graphe (Section 3) ;

– la facilité d'utilisation de la version parallèle d'ILOG Solver (Section 4) ;

– les réglages de la recherche qui nous a permis d'améliorer significativement la qualité des solutions obtenues en un temps de calcul donné (Section 5).

2. Le benchmark

Description du problème

Soit $G = (X, U)$ un graphe orienté, constitué d'un ensemble de n sommets X et d'un ensemble de m arcs U, où chaque arc (i, j) est un couple de sommets distincts.

A ce graphe est associé un ensemble de d demandes. Chaque demande associe à un couple de sommets (p, q) une valeur entière représentant une quantité de flot à transporter de p vers q. Un unique chemin de p vers q doit être choisi pour « router » cette demande. Autrement dit, chaque demande doit être satisfaite en utilisant un et un seul

chemin de transport.[1] Dans la plupart des cas, une et une seule demande (p, q, Dem_{pq}) est associée à tout couple de sommets (p, q), i.e., une seule demande représente l'ensemble des transports à effectuer de p vers q. Le nombre de demandes est alors égal à $n * (n-1)$ et, pour tout couple de sommets (p, q), la quantité Dem_{pq} doit être routée de p à q par un unique chemin. Bien que dans le cas général plusieurs demandes puissent être associées aux mêmes nœuds de départ et d'arrivée p et q (auquel cas ces demandes peuvent être routées par des chemins différents), nous utiliserons par abus de notation un triplet (p, q, Dem_{pq}) pour représenter une telle demande.

Une fonction $Bmax$ associe à chaque demande (p, q, Dem_{pq}) une limite $Bmax_{pq}$ sur le nombre de bonds, i.e., sur le nombre d'arcs utilisés pour router la demande. En particulier, si $Bmax_{pq} = 1$, la demande de p vers q doit être routée directement sur l'arc (p, q).

Une fonction $Tmax$ associe à chaque nœud i une limite $Tmax_i$ sur la quantité de flux traitée par i. Ceci inclut :

– Le trafic au départ de i, i.e., $\sum_{q \neq i} Dem_{iq}$.

– Le trafic arrivant à i, i.e., $\sum_{p \neq i} Dem_{pi}$.

– Le trafic transitant par i, c'est-à-dire la somme des demandes Dem_{pq}, $p \neq i$, $q \neq i$, pour lesquelles le chemin choisi passe par i.

Notons que le trafic transitant par le nœud i n'est compté qu'une fois. La limite $Tmax_i$ ne s'applique donc pas à la somme des flux entrant dans i et sortant de i. Par contre, il est possible de se ramener à une contrainte sur le flux entrant dans i (qui doit être limité à $Tmax_i - \sum_{q \neq i} Dem_{iq}$) ou, de manière équivalente, à une contrainte sur le flux sortant de i (qui doit être limité à $Tmax_i - \sum_{p \neq i} Dem_{pi}$).

Deux fonctions Pin et $Pout$ associent à chaque nœud i un nombre maximal de ports d'entrée (Pin_i) et de sortie ($Pout_i$).

Pour chaque arc (i, j), K_{ij} capacités possibles $Capa^k_{ij}$, $1 \leq k \leq K_{ij}$, sont données. Nous posons par ailleurs $Capa^0_{ij} = 0$. Une et une seule de ces $K_{ij} + 1$ capacités doit être choisie. Cependant, il peut être autorisé de multiplier cette capacité par un entier compris entre une valeur minimale $Wmin^k_{ij}$ et une valeur maximale $Wmax^k_{ij}$ données. Le problème de dimensionnement consiste donc non seulement à choisir pour chaque arc (i, j) une capacité $Capa^k_{ij}$, mais aussi à choisir le coefficient multiplicateur w^k_{ij} retenu. Notons que la capacité $Capa^k_{ij}$ doit nécessairement être retenue dès lors que $Wmin^k_{ij}$ est supérieur ou égal à 1. Il ne peut donc exister qu'un seul k avec $Wmin^k_{ij} > 0$, sinon le problème n'admet pas de solution.

Les choix effectués pour les arcs (i, j) et (j, i) sont liés de la manière suivante : si la k-ième capacité $Capa^k_{ij}$ est retenue avec un coefficient multiplicateur $w^k_{ij} \neq 0$ pour

1. C'est ce qu'on appelle le problème du monoroutage : un seul chemin doit être déterminé pour chaque demande. Ce problème diffère sur ce point du problème de multiflot, qui en est une relaxation.

l'arc (i, j), alors la k-ième capacité $Capa^k_{ji}$ doit être retenue pour l'arc (j, i), avec le même coefficient multiplicateur $w^k_{ji} = w^k_{ij}$.

Une fonction $Cost$ associe à chaque niveau de capacité possible d'un arc (i, j) et de son arc inverse (j, i) une valeur entière $Cost^k_{ij}$, qui représente le coût de la capacité. Ce coût n'est à comptabiliser qu'une fois pour les deux arcs (i, j) et (j, i). Cependant, lorsqu'un coefficient multiplicateur est associé à une capacité, le même coefficient multiplicateur est appliqué au coût.

Une fonction Sec à valeurs booléennes détermine pour chaque demande si le routage de cette demande doit être sécurisé. $Sec_{pq} = 1$ signifie que le routage doit être sécurisé. $Sec_{pq} = 0$ signifie que la sécurisation n'est pas nécessaire. Lorsque le routage doit être sécurisé, il est interdit de passer par un nœud ou par un arc non sécurisé. Pour chaque nœud i, un indicateur de risque $Risk_i$ indique si le nœud est sécurisé ou non. De même, pour chaque arc (i, j) et chaque k, $1 \leq k \leq K_{ij}$, un indicateur de risque $Risk^k_{ij}$ indique si l'arc dans la configuration k est lui aussi sécurisé. Autrement dit, si $Sec_{pq} = 1$, il est interdit de passer par un nœud intermédiaire tel que $Risk_i = 1$ et il est interdit de passer par un lien tel que $Risk^k_{ij} = 1$.[2]

Le problème consiste à determiner le chemin à associer à chaque demande et les capacités à affecter aux arcs de façon à satisfaire toutes les demandes, à assurer que pour tout arc la capacité choisie (pondérée par le coefficient multiplicateur retenu) est supérieure à la somme des quantités transportées par l'arc, et enfin à minimiser la somme des coûts.

Six paramètres booléens sont utilisées pour créer de multiples variantes (plus précisément, $2^6 = 64$ variantes) d'un même jeu de test.

– Le paramètre sec indique que les contraintes de sécurité doivent être prises en compte.

– Le paramètre $nomult$ interdit l'utilisation de multiplicateurs de capacités. Pour tout arc (i, j), deux cas doivent être distingués : si le fichier de données fournit un $Wmin^k_{ij}$ supérieur ou égal à 1, le choix de $Capa^k_{ij}$ avec le multiplicateur $w^k_{ij} = Wmin^k_{ij}$ est imposé ; sinon, le choix de $Capa^k_{ij}$ est libre, mais $w^k_{ij} \leq 1$ est imposé.

– Le paramètre $symdem$ impose la propriété de symétrie des routages : pour toute demande de p à q, s'il existe une demande de q à p, alors les chemins choisis pour router ces deux demandes doivent être symétriques.[3]

– Le paramètre $bmax$ indique que les contraintes de nombre de bonds pour chaque demande doivent être prises en compte. Ces contraintes sont ignorées lorsque $bmax$ n'est pas positionné.

---

2. Le fichier de données ne fournit qu'un seul indicateur $Risk^k_{ij}$ pour les deux arcs (i, j) et (j, i). Autrement dit, on a toujours $Risk^k_{ij} = Risk^k_{ji}$.
3. De même, au cas où il existe plusieurs demandes entre les mêmes sommets p et q, le positionnement de $symdem$ à $vrai$ impose d'agréger ces demandes et de les router sur le même chemin.

– Le paramètre $pmax$ indique que les contraintes de nombre de ports à chaque nœud doivent être prises en compte. Ces contraintes sont ignorées lorsque $pmax$ n'est pas positionné.

– Le paramètre $tmax$ indique que les contraintes de trafic maximal à chaque nœud doivent être prises en compte. Ces contraintes sont ignorées lorsque $tmax$ n'est pas positionné.

Un vecteur de six bits suffit à spécifier une configuration de ces paramètres. Par exemple, le vecteur « 011000 » indique que seuls les paramètres $nomult$ et $symdem$ sont positionnés à $vrai$.

Fichiers de données

Vingt-et-un fichiers de données, répartis en trois séries, sont disponibles sur simple demande. Chaque fichier est repéré par un indicateur de série (A, B ou C) et un entier indiquant le nombre de nœuds du réseau considéré.

La série A comprend les instances de plus petite taille, de 4 à 10 nœuds. Les valeurs optimales des 64 variantes de A04, A05, A06 et A07 sont connues. Par contre, nous ne disposons de solutions prouvées optimales que pour 44 variantes de A08, une variante de A09 et une variante de A10, cette dernière ayant été obtenue en une semaine de temps de calcul avec ILOG CPLEX[CPL 01].

Les séries B et C comprennent des instances de plus grande taille, de 10, 11, 12, 15, 16, 20 et 25 nœuds. Les instances de la série C ont comparativement aux autres peu de choix de capacité par arc. Des solutions prouvées optimales ne sont connues que pour 12 variantes de C10. La plupart des résultats de cet article ont été obtenus avant que l'ensemble du benchmark ne soit constitué. Aussi, nous ne donnerons des résultats que pour la série A et pour les instances à 10, 11 et 12 nœuds des séries B et C, ces dernières étant limitées à 8 variantes, celles où les paramètres sec, $pmax$ et $tmax$ sont fixés à 0.

Chaque fichier de données se compose de $1 + n + m/2 + d$ lignes.

La première ligne comporte trois entiers :

– Le nombre de nœuds n. Les nœuds seront numérotés de 0 à $n - 1$.

– Le nombre d'arcs bidirectionnels $m/2$, compris entre $n - 1$ et $n * (n - 1)/2$.

– Le nombre de demandes d.

Les n lignes suivantes contiennent chacune cinq entiers :

– Un numéro de nœud i.

– La quantité maximale $Tmax_i$ de trafic traité par i. Cette donnée n'est à prendre en compte que si le paramètre booléen $tmax$ est positionné à $vrai$.

– L'indicateur de risque $Risk_i$. Cette donnée n'est à prendre en compte que si le paramètre booléen sec est positionné à $vrai$.

– Le nombre Pin_i de ports d'entrée du nœud i. Cette donnée n'est à prendre en compte que si le paramètre booléen $pmax$ est positionné à $vrai$.

– Le nombre $Pout_i$ de ports de sortie du nœud i. Cette donnée n'est à prendre en compte que si le paramètre booléen $pmax$ est positionné à $vrai$.

Les $m/2$ lignes suivantes contiennent chacune :

– Deux numéros de nœuds i et j, avec $i < j$.

– Le nombre K_{ij} de capacités possibles sur l'arc (i, j).

– $6 * K_{ij}$ entiers donnant successivement pour chaque capacité : la valeur de la capacité $Capa^k_{ij}$, la valeur de la capacité $Capa^k_{ji}$, le coût $Cost^k_{ij}$, le coefficient multiplicateur minimal $Wmin^k_{ij}$, le coefficient multiplicateur maximal $Wmax^k_{ij}$, et l'indicateur de risque $Risk^k_{ij}$.

Les d lignes suivantes fournissent pour chaque demande :

– Le numéro p du nœud de départ.

– Le numéro q du nœud d'arrivée.

– La quantité Dem_{pq} à acheminer de p à q.

– Le nombre de bonds maximal $Bmax_{pq}$ autorisé pour router cette demande. Cette donnée n'est à prendre en compte que si le paramètre booléen $bmax$ est positionné à $vrai$.

– L'indicateur de sécurisation de la demande Sec_{pq}. Cette donnée n'est à prendre en compte que si le paramètre booléen sec est positionné à $vrai$.

Nous donnons ci-dessous à titre d'exemple le contenu du fichier A04.

```
4 6 12
0 256 1   2   2
1 256 0 256 256
2 256 1   2   2
3 256 0 256 256
0 1 3 64 64 6423 0 3 0 128 128 11853 0 1 1 256 256 22779 0 1 0
0 2 3 64 64 5496 0 3 0 128 128  9999 0 1 1 256 256 19071 0 1 0
0 3 3 64 64 3865 0 3 0 128 128  6831 0 1 1 256 256 12829 0 1 0
1 2 3 64 64 4698 0 3 0 128 128  8403 0 1 1 256 256 15879 0 1 0
1 3 3 64 64 5838 0 3 0 128 128 10683 0 1 1 256 256 20439 0 1 0
2 3 3 64 64 4884 0 3 0 128 128  8775 0 1 1 256 256 16623 0 1 0
0 1 65 2 1
1 0 65 2 1
0 2 23 2 0
2 0 23 2 0
0 3 14 2 0
3 0 14 2 0
1 2 42 2 0
2 1 42 2 0
```

```
1 3  7 2 0
3 1  7 2 0
2 3  4 2 0
3 2  4 2 0
```

Cette instance comprend 4 nœuds, $6 * 2$ arcs (graphe complet) et 12 demandes.

Chaque nœud a une capacité de traitement limitée à 256. Les nœuds 0 et 2 sont sécurisés et ne disposent que de deux ports d'entrée et de deux ports de sortie.

Pour chaque arc, 3 capacités sont possibles, de valeurs 64, 128 et 256, dans les deux sens. Les coefficients multiplicateurs possibles vont de 0 à 3 pour la capacité 64 et de 0 à 1 pour les capacités 128 et 256. Les capacités égales à 128 sont sécurisées. Seuls les coûts diffèrent selon les arcs.

Chaque demande doit être routée par un chemin de longueur inférieure ou égale à 2 arcs (si $bmax$ est $vrai$). Les communications entre les nœuds 0 et 1 doivent être sécurisées (si sec est $vrai$)

Notons que sur cette instance il n'est jamais optimal de multiplier la capacité 64 par 2 (sauf pour sécuriser), puisque le coût d'une capacité de 128 est toujours inférieur au double du coût d'une capacité de 64. Par contre, le triple du coût de la capacité 64 est inférieur au coût de la capacité 256, d'où un intérêt potentiel à utiliser le multiplicateur pour introduire une capacité de 192. Cette possibilité n'est envisageable que si la paramètre $nomult$ est positionné à $faux$.

Remarques

D'autres versions de la contrainte de sécurité pourront être envisagées dans le futur. Telle que nous l'avons définie dans le présent benchmark, la contrainte de sécurité est particulièrement forte : les demandes pour lesquelles $Sec_{pq} = 1$ peuvent être acheminées sur leur route nominale même si tous les nœuds et tous les arcs sécurisés tombent simultanément en panne. Une variante plus réaliste consisterait à définir pour tout risque de panne non négligeable (considéré individuellement) des routages de remplacement permettant d'acheminer les demandes telles que $Sec_{pq} = 1$. La formulation exacte du problème serait néanmoins beaucoup plus complexe. C'est pourquoi nous nous sommes limités, pour la constitution du présent benchmark, à une version plus simple, mais moins réaliste, de la contrainte de sécurité.

Notons également qu'une autre version du benchmark, plus proche du problème de multi-flot, pourrait être définie en relachant la contrainte de mono-routage, i.e., en autorisant le flux d'un nœud p à un nœud q à se répartir en plusieurs chemins. La contrainte de symétrie impose alors aux flux de p à q et de q à p d'emprunter les mêmes chemins dans les mêmes proportions, tandis que la contrainte sur le nombre de bonds s'applique à tous les chemins utilisés.

Notons enfin que le problème du dimensionnement de réseau avec mono-routage a été peu étudié. Rothlauf, Goldberg et Heinzl [ROT 02] ont travaillé sur des problèmes similaires à 15, 16 et 26 nœuds, fournis par Deutsche Telekom, mais en obligeant le

réseau solution à être un arbre. Gabrel, Knippel et Minoux [GAB 99] ont développé une méthode exacte pour résoudre des problèmes de dimensionnement de réseaux de 8 à 20 nœuds avec fonctions de coût en escalier, mais en considérant un routage de type multiflot. De fait, les problèmes avec multiflot ont été les plus étudiés, avec divers types de fonctions de coût, par exemple un coût fixe et un coût proportionnel au trafic comme dans [GEN 94].

Enfin, toutes le variantes doivent être résolues dans un temps fixe de 10 minutes.

3. Programmation par contraintes et graphes

Comme indiqué précédemment, le formalisme de la programation par contraintes nous a paru particulièrement adapté au problème de l'ajout de contraintes additionnelles. Une utilisation naïve d'un outil de programmation par contraintes comme ILOG Solver ne permet néanmoins pas de résoudre le problème dans des conditions d'efficacité satisfaisantes : les premières expériences menées par France Télécom R&D ont montré qu'à partir de 8 nœuds, il pouvait être difficile de générer des solutions à moins de 20% de l'optimum en un temps de calcul limité à quelques minutes. Il nous a donc paru nécessaire de tirer plus amplement parti de la structure du problème, et notamment du fait qu'il s'agit, pour l'essentiel, de choisir des chemins pour router des communications dans un graphe.

Nous avons donc en quelque sorte introduit un nouveau type de variable représentant un chemin d'un nœud « source » donné vers un nœud « puits » donné. Plus précisément, un chemin est défini par deux variables ensemblistes, représentant l'ensemble des nœuds et l'ensemble des arcs du chemin, et un ensemble de contraintes entre ces deux variables ensemblistes :

– Si un arc appartient au chemin, ses deux extrémités appartiennent au chemin.

– Un et un seul arc sortant de la source du chemin doit appartenir au chemin.

– Un et un seul arc entrant dans le puits du chemin doit appartenir au chemin.

– Si un nœud différent de la source et du puits appartient au chemin, alors un et un seul arc entrant dans ce nœud et un et un seul arc sortant de ce nœud doivent appartenir au chemin.

Plusieurs contraintes globales ont par ailleurs été implantées de manière à détecter les nœuds et les arcs devant appartenir à un chemin donné (pour des raisons de connexité), éliminer les nœuds et les arcs ne pouvant pas appartenir à un chemin donné, raisonner sur le nombre de nœuds et d'arcs d'un chemin, et relier les « variables chemins » ainsi définies aux variables définissant la capacité et le niveau de sécurité (risqué ou non) des nœuds et des arcs du réseau.

La plus importante de ces contraintes globales identifie les nœuds et les arcs par lesquels un chemin doit impérativement passer, en tenant compte des informations disponibles sur le nombre maximal $Bmax$ d'arcs du chemin. L'algorithme utilisé pour propager cette contrainte est le suivant :

– Utilisation de l'algorithme de Ford (cf. [GON 95]) pour identifier le plus court chemin admissible (en nombre d'arcs) entre la source et chaque nœud du graphe ;

– Utilisation de l'algorithme de Ford pour identifier le plus court chemin admissible (en nombre d'arcs) entre chaque nœud du graphe et le puits ;

– Utilisation des longueurs de chemins ainsi calculées pour éliminer les nœuds par lesquels ne peut passer aucun chemin de longueur inférieure ou égale à $Bmax$ arcs ;

– Utilisation des longueurs de chemins ainsi calculées pour marquer les nœuds qui peuvent trivialement être contournés par un chemin de longueur inférieure ou égale à $Bmax$ arcs ;

– Utilisation pour chaque nœud non marqué de l'algorithme de Ford dans le but de déterminer s'il existe un chemin de la source au puits de longueur inférieure ou égale à $Bmax$ arcs ne passant pas par le nœud considéré.

L'utilisation de cet algorithme s'est avérée globalement rentable, malgré sa complexité en O($n * m * Bmax$), soit O(n^4) dans le pire des cas.

Nous avons par ailleurs utilisé un algorithme de plus court chemin en tant qu'heuristique pour guider la recherche de bonnes solutions. La stratégie utilisée est extrêmement simple. A chaque étape de la résolution, on choisit le chemin non instancié pour lequel l'expression $2 * dem_{pq} + dem_{qp}$ est la plus forte. Ceci permet de pondérer l'importance donnée à une demande de p vers q par le poids de la demande inverse qui, même lorsque le paramètre $symdem$ n'est pas positionné, a dans la solution optimale de fortes chances de suivre un chemin de q vers p symétrique de celui suivi de p de q. Nous déterminons alors le chemin le moins coûteux (marginalement compte-tenu des demandes déjà routées) pour router cette demande. Un point de choix est alors créé : dans la première branche, la demande est contrainte à passer par le dernier arc de ce chemin qui n'est pas encore imposé ; en cas de backtrack, cet arc est au contraire interdit pour cette demande. Lorsqu'un chemin est totalement instancié, nous passons à la demande suivante. Une nouvelle solution est obtenue lorsque tous les chemins sont instanciés. La recherche continue alors en contraignant le coût des nouvelles solutions à être strictement inférieur au coût de la meilleure solution déjà trouvée.

Pour favoriser l'obtention rapide de bonnes solutions, nous avons encapsulé cette procédure de résolution dans un mécanisme de « Slice-Based Search » (SBS), une implantation du « Discrepancy Bounded Depth First Search » [BEC 00] dans lequel un maximum de recalculs sont évités.

Le tableau ci-dessous résume les résultats obtenus. Pour chaque instance, nous sommons les valeurs obtenues pour chaque paramétrage après un maximum de 10 minutes de temps de calcul sur un PC Pentium III à 700 MHz. Comme indiqué précédemment, 64 paramétrages sont retenus pour les instances de la série A et 8 pour les instances des séries B et C. Le tableau fournit pour chaque instance, la somme des meilleures bornes inférieures connues (lorsque ce nombre est significatif), la somme des coûts des meilleures solutions connues, la somme des coûts des solutions trouvées par notre algorithme (PPC) et la somme des coûts des solutions trouvées par deux autres algorithmes à base de programmation linéaire en nombre entiers (une variante

du modèle MIP précédent) et de génération de colonnes (GC), toujours avec une limite de 10 minutes de temps de calcul. Un « fail » dans le tableau signifie qu'il existe au moins un paramètrage pour lequel l'algorithme ne donne aucune solution en 10 minutes. Le tableau fournit également pour chaque algorithme l'écart relatif moyen aux meilleures solutions connues. Notons que quand la taille du problème augmente, le MIP ne parvient pas toujours à générer au moins une solution entière. L'algorithme de PPC est le plus robuste. Cependant, sur les séries B et C, les résultats ne sont pas toujours satisfaisants en terme d'écart. En particulier, sur B11, B12 et C12, la PPC ne fournit pas les meilleurs résultats en moyenne.

Instance	Σ BI	Σ BS	Σ PPC	Ecart PPC	Σ MIP	Ecart MIP	Σ GC	Ecart GC
A04	1782558	1782558	1782558	0.00%	1782558	0.00%	1782558	0.00%
A05	2351778	2351778	2351778	0.00%	2351778	0.00%	2351778	0.00%
A06	2708264	2708264	2708264	0.00%	2708264	0.00%	2708264	0.00%
A07	3290590	3290590	3290940	0.01%	3337284	1.42%	3310007	0.60%
A08	4002083	4050151	4076785	0.69%	4417611	9.06%	4263830	5.11%
A09	4374737	4966858	5027246	1.25%	5934187	19.44%	5621264	12.85%
A10	4958292	5843478	5934297	1.57%	fail	fail	fail	fail
B10		155748	172255	10.62%	174494	12.04%	176625	13.40%
B11		271080	320356	19.20%	302226	12.46%	300317	11.70%
B12		207424	235412	13.49%	235031	13.32%	227387	9.62%
C10		102824	104686	1.84%	106183	3.24%	105578	2.72%
C11		169124	179078	5.90%	184485	9.11%	199265	17.83%
C12		310788	360673	16.20%	fail	fail	348666	12.26%

Une étude des paramétrages posant le plus de difficulté à chacun de ces algorithmes montre que comparativement :

– les contraintes qui posent le plus de difficultés à l'algorithme de génération de colonnes sont les contraintes de sécurité (sec), de nombre de ports maximal ($pmax$) et de trafic maximal ($tmax$) ; par contre, la présence de la contrainte de nombre de bonds ($bmax$) est un élément favorable ;

– la contrainte qui pose le plus de difficultés à l'algorithme de programmation par contraintes est la contrainte de trafic maximal ;

– les contraintes qui posent le plus de difficultés au MIP sont les contrainte de nombre de bonds et de trafic maximal ;

– l'algorithme de programmation par contraintes semble tirer comparativement moins bien parti que les deux autres de la contrainte de chemins symétriques ($symdem$), alors que dans les trois cas le nombre de chemins est divisé par deux.

4. Parallélisation

ILOG Parallel Solver[PER 99] est un produit inclus dans Ilog Solver. Il implante un parallélisme-*ou* basé sur la notion de recalcul d'états[KER 89]. Il est basé sur un paradigme de mémoire partagée et n'utilise pas la notion d'ordinateurs distribués.

Il procède comme suit : Chaque agent se partage le même modèle du problème et possède chacun sa propre copie des objets de calcul. Pendant la recherche de solution,

ils se partagent et explorent le même arbre de recherche. Enfin, en cas de famine, un mécanisme de balancement de charge s'assure qu'aucun agent ne reste sans travail alors qu'il reste des portions de l'arbre à explorer. Finalement, en fin de recherche, tous les agents se synchronisent et terminent la recherche en même temps. Pour plus de détails, se reporter à [PER 99].

Le passage de ILOG Solver à ILOG Parallel Solver ne requiert que l'ajout de quelques lignes au prototype initial. Le tableau ci-dessous résume les résultats obtenus sur les séries B et C avec ILOG Parallel Solver. Ces résultats ont été obtenus avec la même limite de 10 minutes sur un quadri-pentium III Xeon à 700 Mhz. Dans la moitié des cas, on trouve avec ILOG Parallel Solver en 2 minutes 30 des solutions meilleures que la version séquentielle en 10 minutes. Enfin, le gain de la parallélisation est net.

Instance	B10	B11	B12	C10	C11	C12	Moyenne
Ecart Solver	10.62%	19.20%	13.49%	1.84%	5.90%	16.20%	**11.20%**
Ecart //Solver	6.32%	13.12%	11.86%	0.99%	3.85%	13.32%	**8.24%**

5. Réglage de la recherche

Nous avons implanté deux variations par rapport à la procédure de recherche DBDFS[BEC 00]. La procédure originale implante le méchanisme suivant : Les nœuds ouverts de l'arbre sont rangés dans une queue de priorité avec leur évaluation. Cette évaluation est égale au résultat de la division entière du nombre de déplacements à droite que comporte le chemin de la racine de l'arbre jusqu'au nœud à evaluer, par la largeur donnée en paramètre à l'évaluateur de nœuds.

En utilisant le mécanisme de trace de ILOG Solver, nous avons constaté que pour de nombreuses bonnes solutions, les déplacements à droite se trouvaient en haut de l'arbre. Or les demandes étant traités par ordre décroissant de poids, ils correspondent donc aux demandes prépondérantes pour le coût total. Nous avons donc implanté une évaluation qui dépend de la profondeur totale de l'arbre. Dans tous les cas, on calcule une somme de poids d'un nœud et on retourne le résultat divisé par 2 et arrondi à l'entier inférieur.

Dans le premier cas, le poids d'un nœud est la somme des poids de chaque mouvement vers la droite dans le chemin qui relie le sommet de l'arbre au nœud. Le poids d'un mouvement droit est égal à 1 si on est dans le premier cinquième de l'arbre, 4 dans la partie entre le cinquième et la moitié haute de l'arbre et ∞ dans la moitié inférieure.[4]

Dans le deuxième cas, le poids d'un déplacement vers la droite dépend de la profondeur totale de l'arbre. Il croit avec la profondeur du nœud. Mais il s'adapte selon la profondeur totale pour essayer de toujours remettre en cause les décisions prises en haut de l'arbre. Par exemple, si la profondeur totale est inférieure à 60, un mouvement vers la droite pèse 1 dans le premier tiers de l'arbre, 4 dans le deuxième tiers et ∞

4. Ce qui correspond effectivement à ne pas considérer les nœuds ayant un déplacement vers la droite dans la moitié inférieure de l'arbre.

dans le dernier tiers. Par contre, si la profondeur de l'arbre est supérieure à 240, un mouvement vers la droite pèse 1 dans le premier septième de l'arbre, puis 3 jusqu'à au cinquième de l'arbre, 6 jusqu'au tiers et ∞ en dessous. Le tableau suivant résume les résultats obtenus avec ILOG Solver et ILOG Parallel Solver pour ces deux variantes.

Instance	B10	B11	B12	C10	C11	C12	Moyenne
Var 1, Séq	8.18%	13.72%	11.57%	2.01%	6.58%	15.64%	**9.60%**
Var 1, //	3.10%	10.36%	11.43%	1.70%	2.98%	11.97%	**6.92%**
Var 2, Séq	6.91%	13.85%	12.41%	1.92%	6.42%	14.89%	**9.40%**
Var 2, //	3.50%	7.68%	11.81%	1.73%	3.66%	12.99%	**6.89%**

6. Conclusion

Dans cet article, nous avons présenté un benchmark original, permettant de tester la robustesse d'un algorithme vis à vis de l'ajout de nouvelles contraintes, avec pour objectif d'obtenir les meilleures performances en moyenne. Nos premières expériences montrrent que des résultats satisfaisants en moyenne peuvent être obtenus en intégrant des algorithmes de graphes au sein d'un outil de programmation par contraintes comme ILOG Solver. De plus, les résultats obtenus ont pu être significativement améliorés, d'une part en passant de ILOG Solver à ILOG Parallel Solver, d'autre part en répartissant plus finement l'effort de recherche.

Ces résultats restent néanmoins fortement améliorables, comme le montrent certains écarts moyens, supérieurs à 10%. De nombreuses idées, complémentaires ou concurrentes à notre approche actuelle, sont susceptibles de donner lieu à de nouveaux travaux de recherche : renforcement de la propagation des contraintes, notamment des contraintes portant directement sur le coût de la solution ; répartition adaptative de l'effort de recherche entre le haut et le bas de l'arbre de recherche ; utilisation d'algorithmes d'optimisation locale ou évolutionnaires ; amélioration et parallélisation de notre algorithme de génération de colonnes ; amélioration par des heuristiques ou des coupes et parallélisation de notre algorithme de programmation linéaire en nombres entiers ; définition d'une stratégie hybride donnant plus ou moins d'importance à chaque technique de base en fonction de la taille du problème et des contraintes considérées.

Comme nous l'avons déjà indiqué, les fichiers de données sont à la disposition de la communauté scientifique, ainsi que les différents éléments permettant de tester de nouveaux algorithmes et de les comparer aux résultats que nous avons présentés.

7. Remerciements

Les travaux présentés dans cet article ont été financés en partie par le MENRT, dans le cadre du projet RNRT ROCOCO. Nous tenons particulièrement à remercier nos partenaires, notamment Dominique Barth du laboratoire PRiSM et Claude Lemaréchal de l'INRIA Rhône-Alpes, ainsi que Christian de Sainte-Marie, Alain Chabrier et Philippe Réfalo pour leurs contributions au projet.

Nous tenons tout particulièrement à remercier la société Intel pour le prêt du quadri-processeur qui nous a permis de mener à bien ces expériences réussies sur la parallélisation de la résolution de ce problème de conception de réseau.

8. Bibliographie

[BEC 00] BECK J. C., PERRON L., « Discrepancy-Bounded Depth First Search », *Proceedings of CP-AI-OR 00*, March 2000.

[CPL 01] CPLEX, « ILOG CPLEX 7.5 User's Manual and Reference Manual », ILOG, S.A., 2001.

[GAB 99] GABREL V., KNIPPEL A., MINOUX M., « Exact Solution of Multicommodity Network Optimization Problems with General Step Cost Functions », *Operations Research Letters*, vol. 25, 1999, p. 15-23.

[GEN 94] GENDRON B., CRAINIC T. G., « Relaxations for Multicommodity Capacitated Network Design Problems », rapport, 1994, Centre de Recherche sur les Transports, Université de Montréal.

[GON 95] GONDRAN M., MINOUX M., *Graphes et algorithmes*, Eyrolles, 1995.

[KER 89] CHASSIN DE KERGOMMEAUX J., CODOGNET P., ROBERT P., SYRE J.-C., « Une programmation logique parallèle : Systèmes non-déterministes », *TSI*, vol. 04, 1989, p. 285-305.

[PER 99] PERRON L., « Search procedures and parallelism in constraint programming », JAFFAR J., Ed., *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP99)*, Springer-Verlag, 1999, p. 346-360.

[ROT 02] ROTHLAUF F., GOLDBERG D. E., HEINZL A., « Network Random Keys : A Tree Representation Scheme for Genetic and Evolutionary Algorithms », *Evolutionary Computation*, vol. 10, n° 1, 2002, p. 75-97.

[SOL 01] SOLVER, « ILOG Solver 5.2 User's Manual and Reference Manual », ILOG, S.A., 2001.

9. Annexe : Une formulation linéaire en nombres entiers

Une formulation linéaire simple du problème introduit une variable binaire x_{ij}^{pq} pour chaque demande (p, q, Dem_{pq}) et chaque arc $(i, j) \in U$. Cette variable prend la valeur 1 si l'arc (i, j) est utilisé pour transmettre la demande de p à q, et prend la valeur 0 sinon.

Pour chaque arc $(i, j) \in U$, on introduit K_{ij} variables entières w_{ij}^k, $1 \leq k \leq K_{ij}$. La variable w_{ij}^k désigne le multiplicateur associé à la k-ième capacité possible de l'arc (i, j). Pour i et j donnés, un seul w_{ij}^k peut être différent de 0. Pour modéliser cette contrainte, on introduit une variable booléenne y_{ij}^k qui vaut 1 si c'est la k-ième capacité qui est choisie, et 0 sinon.[5]

5. Notons que, lorsque $pmax$ est $faux$, les variables w_{ij}^k peuvent être supprimées en introduisant toutes les capacités intermédiaires susceptibles d'être obtenues par multiplication et en supprimant toutes les combinaisons dominées en termes de coût (et de sécurité). Ainsi, dans l'exemple précédent, il suffirait d'introduire la capacité 192 pour chaque arc. L'intérêt d'une

Les contraintes communes à toutes les versions du problème sont alors les suivantes :

– Toute demande (p, q, Dem_{pq}) part de son site de départ : $\sum_{j:(p,j)\in U} x_{pj}^{pq} = 1$

– Toute demande (p, q, Dem_{pq}) finit à son site d'arrivée : $\sum_{i:(i,q)\in U} x_{iq}^{pq} = 1$

– Toute demande (p, q, Dem_{pq}) suit un chemin unique :
$\forall (i,j) \in U, x_{ij}^{pq} \in \{0,1\}$
$\forall i \in \{1,\ldots,n\}, x_{ip}^{pq} = 0$
$\forall i \in \{1,\ldots,n\}, x_{qi}^{pq} = 0$
$\forall i \in \{1,\ldots,n\}, i \neq p, i \neq q, \sum_{j:(i,j)\in U} x_{ij}^{pq} = \sum_{j:(j,i)\in U} x_{ji}^{pq}$

– Pour tout arc (i, j), une seule capacité peut être choisie :
$\forall k \in \{1,\ldots,K_{ij}\}, y_{ij}^k \in \{0,1\} \sum_{k\in\{1,\ldots,K_{ij}\}} y_{ij}^k \leq 1$

– Pour tout arc (i, j), un et un seul coefficient multiplicateur w_{ij}^k est choisi non nul entre $Wmin_{ij}^k$ et $Wmax_{ij}^k$:
$\forall k \in \{1,\ldots,K_{ij}\}, w_{ij}^k \in \{Wmin_{ij}^k, Wmin_{ij}^k + 1, \ldots, Wmax_{ij}^k\}$
$\forall k \in \{1,\ldots,K_{ij}\}, y_{ij}^k \leq w_{ij}^k \leq Wmax_{ij}^k y_{ij}^k$

– Pour tout arc (i, j), les capacités installées sur les arcs (i, j) et (j, i) sont cohérentes : $\forall k \in \{1,\ldots,K_{ij}\}, w_{ij}^k = w_{ji}^k$

– La demande totale routée par tout arc (i, j) est inférieure ou égale à sa capacité :
$\sum_{p,q\in\{1,\ldots,n\}} Dem_{pq} x_{ij}^{pq} \leq \sum_{k\in\{1,\ldots,K_{ij}\}} Capa_{ij}^k w_{ij}^k$

Le coût est défini comme suit : $\sum_{(i,j)\in U: i<j} \sum_{k\in\{1,\ldots,K_{ij}\}} Cost_{ij}^k w_{ij}^k$

Les contraintes additionnelles ci-dessous sont à ajouter selon la valeur des paramètres :

– Si sec est $vrai$: pour toute demande (p, q, Dem_{pq}) telle que $Sec_{pq} = 1$ et pour tout nœud $i \neq p$ tel que $Risk_i = 1$, ajouter la contrainte $\sum_{j:(i,j)\in U} x_{ij}^{pq} = 0$. Cette contrainte interdit de router la demande par le nœud intermédiaire i.

– Si sec est $vrai$: pour toute demande (p, q, Dem_{pq}) telle que $Sec_{pq} = 1$ et pour tout arc (i, j), ajouter la contrainte $x_{ij}^{pq} + \sum_{k:Risk_{ij}^k=1} y_{ij}^k \leq 1$. Cette contrainte interdit de router la demande par l'arc (i, j) si la capacité choisie sur l'arc (i, j) n'est pas sécurisée.

– Si $nomult$ est $vrai$: pour tout arc (i, j), s'il existe $k \in \{1,\ldots,K_{ij}\}$ tel que $Wmin_{ij}^k \geq 1$, ajouter la contrainte $w_{ij}^k = Wmin_{ij}^k$; sinon, ajouter la contrainte $\sum_{k\in\{1,\ldots,K_{ij}\}} w_{ij}^k \leq 1$. Notons que les variables y_{ij}^k peuvent alors être supprimées (sauf si sec est positionné à $vrai$).

– Si $symdem$ est $vrai$: pour tout arc (i, j) et toute paire de demandes (p, q, Dem_{pq}) et (q, p, Dem_{qp}), ajouter la contrainte $x_{ij}^{pq} = x_{ji}^{qp}$.

– Si $bmax$ est $vrai$: pour toute demande (p, q, Dem_{pq}), ajouter la contrainte $\sum_{(i,j)\in U} x_{ij}^{pq} \leq Bmax_{pq}$.

– Si $pmax$ est $vrai$: pour tout nœud i, ajouter les contraintes
$\sum_{j:(j,i)\in U} \sum_{k:Capa_{ji}^k\neq 0} w_{ji}^k \leq Pin_i$ et $\sum_{j:(i,j)\in U} \sum_{k:Capa_{ij}^k\neq 0} w_{ij}^k \leq Pout_i$.

– Si $tmax$ est $vrai$: pour tout nœud i, ajouter la contrainte
$\sum_{p\neq i} \sum_{q\neq i} \sum_{j:(i,j)\in U} Dem_{pq} x_{ij}^{pq} + \sum_{p\neq i} Dem_{pi} + \sum_{q\neq i} Dem_{iq} \leq Tmax_i$.

---

telle transformation du problème dépend évidemment du nombre de valeurs additionnelles introduites.