Une approche simple pour un problème de réseaux complexe

Diego Olivier Fernandez Pons

Une approche simple pour un problème de réseaux complexe

Diego Olivier Fernandez Pons [1 2]

[1] Université Pierre et Marie Curie - Paris VI, 5 place Jussieu 75005 Paris
[2] ILOG S.A, 9 rue de Verdun 94253 Gentilly Cedex
dofpons@ilog.fr

Résumé

La programmation par contraintes a réputation d'être difficile à mettre en oeuvre dans les applications réelles. ILOG a d'ailleurs utilisé des méthodes fort complexes pour résoudre un problème réel de dimensionnement de réseaux : méthodes hybrides, variables et contraintes dédiées, heuristiques basées sur des algorithmes de recherche opérationnelle, apprentissage, etc. Ce travail étudie ce même problème de réseaux avec d'avantage de contraintes réelles (séparation entre réseau dorsal et réseau local) et pourtant n'utilise que des moyens élémentaires. Nous montrerons comment exploiter la consistance forte et la recherche par défaut d'ILOG CP Optimizer pour détecter les faiblesses du modèle et y remédier. Nous aboutissons ainsi à une solution de qualité acceptable et simple à mettre en oeuvre. Nous proposons aussi des directions à étudier susceptibles d'améliorer le solveur existant.

Abstract

Constraint programming is said to be complex to use in industrial applications. For instance, ILOG developed complex constraint programming based techniques for a network design problem : hybrid solvers, dedicated variables, search heuristics based on operations research algorithms, learning, etc. In this work we study the same network design problem with even more realistic constraints (we have added a backbone/star constraint) but using only simple constraint programming techniques. We will show how to use strong consistency and default search provided by ILOG CP Optimizer to find the model weaknesses and solve them. We end with a very simple solution of enough quality and some research directions with a fair insurance that they will conduce to improvements.

1 Introduction

La complexité des méthodes de programmation par contraintes mises en oeuvre dans les problèmes réels peut donner l'impression qu'il s'agit d'une technique complexe à déployer en contexte industriel. Ainsi le seul problème de dimensionnement de réseaux monoroute a donné lieu à une multitude d'approches basées sur la programmation par contraintes, souvent hybrides (recherche locale, programmation linéaire) dans tous les cas associées à des algorithmes dédiés (heuristiques, algorithmes de propagation, bornes inférieures). On est bien loin de l'idéal déclaratif des origines de la programmation par contraintes.

Le propos de ce travail est d'exploiter au mieux les fonctionnalités de haut niveau des outils de programmation par contraintes et d'interpréter les informations ainsi obtenues pour déterminer les éléments critiques du problème de dimensionnement de réseaux, afin d'y apporter une réponse adaptée (éventuellement dédiée). En effet, le développement d'un algorithme dédié est une procédure longue et risquée. Il est donc désirable d'une part de minimiser leur utilisation, d'autre part d'avoir une assurance qu'une procédure spécifique va conduire à un gain réel de performances. On évite ainsi l'erreur courante de spécialisation précoce de la méthode d'optimisation laquelle peut compromettre l'évolution de la solution.

A cette fin l'utilisation d'ILOG CP Optimizer dans cette étude est limitée à la couche de modélisation (pas de contraintes dédiées) et à sa procédure de recherche prédéfinie, éventuellement paramétrée par des phases de recherche.

2 Description du problème de dimensionnement de réseaux

Nous décrivons d'abord le problème Rococo et les méthodes utilisées précédemment pour le résoudre.

2.1 Problème de base (Rococo)

Le problème Rococo consiste à choisir pour tout arc (i, j) d'un graphe G un type de lien $y_{ij} \in [0 \dots k]$ parmi un nombre fixe de possibilités chacune étant caractérisée par un coût, une capacité maximale et éventuellement d'autres propriétés selon la variante du problème choisie.

Cet ensemble de liens forme un réseau dans lequel il faut router les communications qui doivent s'établir entre chaque couple de points.

Toutes les demandes du problème doivent pouvoir être routées simultanément dans le réseau construit, chaque route étant un chemin reliant la source s^d et la destination t^d, éventuellement soumis à des contraintes additionnelles. Chaque demande d a un volume Vol_d et ce volume s'additionne avec celui des autres demandes dans chaque arc qu'elle emprunte. La variable $x_{ij}^d \in \{0, 1\}$ indique si la demande d emprunte l'arc (i, j). L'objectif du problème est de minimiser le coût de construction du réseau.

Le problème peut être décrit par le programme mathématique schématique suivant :

$$\min \sum_{ij \in G} \mathsf{Cout}\,[y_{ij}]$$

$$\forall ij \in G,\ \sum_d \mathsf{Vol}_d x_{ij}^d \ \leq\ \mathsf{Capacite}\,[y_{ij}] \qquad (1)$$

$$\forall d,\ (x_{ij}^d)_{ij \in G} \text{ chemin entre } s^d \text{ et } t^d \qquad (2)$$

$$\text{contraintes supplementaires} \qquad (3)$$

$$x_{ij}^d \in \{0, 1\} \quad y_{ij} \in [0 \dots k]$$

Le problème Rococo comporte 6 contraintes optionnelles. Chaque problème est identifié par un vecteur de booléens qui indique les contraintes qui sont actives.

sécurité Certaines communications ont des besoins de tolérance aux pannes plus importants. Ces demandes dites sécurisées ne peuvent être routées que par des liens dits eux-mêmes sécurisés et qui sont beaucoup plus onéreux à capacité égale. Les demandes ordinaires peuvent utiliser indifféremment des liens sécurisés ou pas.

Si l'on note s_{ij} une variable qui détermine si le lien installé sur l'arc (i, j) doit être sécurisé alors la contrainte s'écrit $\forall d \in \mathsf{Sec},\ [x_{ij}^d = 1] \Rightarrow [s_{ij} = 1]$

multiplicité Un lien peut être installé plusieurs fois sur un arc. Si les capacités disponibles sont 64, 128 et 256, on peut par exemple installer 3 liens de 64 unités, obtenant un lien pouvant véhiculer 192 unités.

Techniquement parlant cela revient à ajouter une propriété $m_{ij} \in \mathbb{N}$ pour chaque lien et à remplacer y_{ij} par $m_{ij} \times y_{ij}$ dans les contraintes. Les bornes sur la multiplicité des liens dépendent du type de lien autrement dit $\forall ij \in G,\ \mathsf{MinMult}[y_{ij}] \leq m_{ij} \leq \mathsf{MaxMult}[y_{ij}]$. Quand la multiplication des liens est interdite, on ajoute la contrainte $m_{ij} \leq 1$.

symétrie On peut imposer que toutes les communications allant de s à t et de t à s utilisent le même chemin. Cette contrainte s'écrit $\forall pq,\ [s^p = t^q \wedge s^q = t^p] \Rightarrow [\forall ij,\ x_{ij}^p = x_{ji}^q]$

longueur des chemins La longueur des chemins peut être limitée. Cette contrainte s'écrit $\sum_{ij} x_{ij}^d \leq \mathsf{LongueurMax}^d$

limitation de ports Le nombre de liens entrants et sortant d'un noeuds peut être limité. Cette limitation prend en compte l'éventuelle multiplicité m_{ij} des liens. La contrainte s'écrit donc $\sum_j m_{ij} \leq \mathsf{PortsMax}_i$

limitation de trafic On peut imposer une limite au trafic circulant dans un noeud. Si l'on introduit une variable x_i^d indiquant que la demande d passe par le noeud i, la contrainte de trafic s'écrit $\forall i,\ \sum_d \mathsf{Vol}^d x_i^d \leq \mathsf{TraficMax}_i$

Enfin, en première approximation les problèmes Rococo ont des liens symétriques : le même type de lien doit être installé sur les arcs (i, j) et (j, i). On peut au choix considérer que le graphe G est non orienté et qu'un lien possède une capacité directe $i \to j$ et une capacité inverse $j \to i$, ou bien considérer que le graphe G est orienté et ajouter les contraintes $y_{ij} = y_{ji}$ et $m_{ij} = m_{ji}$.

Un modèle complet du problème Rococo est présenté dans la table 1.

2.2 Approches précédentes en PPC du problème de dimensionnement

Le problème de dimensionnement de réseaux avec routage par chemins et contraintes additionnelles a fait l'objet de plusieurs études en programmation par contraintes. Elles ont toutes en commun la grande technicité dans les méthodes utilisées :

Jeannin et Givry [9] ont utilisé une méthode à grands voisinages semblable à Perron [15] pour un problème proche au sein du projet EOLE regroupant Cosytec, Thalès, France Télécom, Bouygues, l'ONERA et l'École des Mines de Nantes. Il convient de noter

$$\min \sum_{ij \in G} \mathsf{Cout}\,[y_{ij}]$$

$$\forall ij \in G,\ \sum_d \mathsf{Vol}^d x_{ij}^d \ \leq\ m_{ij} \times \mathsf{Capacite}\,[y_{ij}] \quad (4)$$

$$\begin{cases} \forall d,\ \forall j \neq s^d, t^d,\ \sum_i x_{ij}^d = x_j^d = \sum_k x_{jk}^d \\ \forall d,\ x_{s^d} = 1 \wedge \sum_i x_{is} = 0 \\ \forall d,\ x_{t^d} = 1 \wedge \sum_j x_{tj} = 0 \end{cases} \quad (5)$$

$$\begin{cases} \forall ij \in G,\ y_{ij} = y_{ji} \\ \forall ij \in G,\ m_{ij} = m_{ji} \end{cases} \quad (6)$$

$$\forall ij \in G,\ \mathsf{MinMult}[y_{ij}] \leq m_{ij} \leq \mathsf{MaxMult}[y_{ij}] \quad (7)$$

$$\forall d \in \mathsf{Sec},\ [x_{ij}^d = 1] \Rightarrow [s_{ij} = 1] \quad (8)$$

$$\forall ij \in G,\ m_{ij}\ \leq\ 1 \quad (9)$$

$$\forall pq,\ [s^p = t^q \wedge s^q = t^p] \Rightarrow [\forall ij,\ x_{ij}^p = x_{ji}^q] \quad (10)$$

$$\forall d,\ \sum_{ij} x_{ij}^d\ \leq\ \mathsf{LongueurMax}^d \quad (11)$$

$$\forall i,\ \sum_j m_{ij} \leq \mathsf{PortsMax}_i \quad (12)$$

$$\forall i,\ \sum_d \mathsf{Vol}^d x_i^d\ \leq\ \mathsf{TraficMax}_i \quad (13)$$

$$x_{ij}^d \in \{0,1\} \quad x_i^d \in \{0,1\} \quad y_{ij} \in [0 \ldots k] \quad m_{ij} \in \mathbb{N}$$

Tab. 1 – Modèle complet du problème Rococo. Les contraintes (4) et (5) correspondent aux contraintes (1) et (2) du modèle simplifié. La contrainte (6) est une façon d'exprimer que le graphe G n'est pas orienté. La contrainte (7) ajuste les bornes des multiplicateurs. Les contraintes (8-13) sont les contraintes optionnelles.

qu'ils ont à cette fin utilisé la bibliothèque de haut niveau ToOLS laquelle permet de décrire les procédures de recherche d'une façon plus déclarative.

Cambazard [4] a testé sur le problème Rococo des approches où la programmation par contraintes est guidée par une borne inférieure calculée par programmation linéaire ou relaxation lagrangienne.

Les techniques mises en oeuvre par Le Pape et al. [2] [12] [6] se répartissent entre modifications du moteur de contraintes, heuristiques dédiées et corrections adhoc des défauts mis en évidence par les jeux de tests.

Modifications du moteur de contraintes :

- Des variables spécialisées pour les graphes qui gèrent les arcs, les noeuds, les origines, les destinations et l'accumulation de grandeurs le long des arcs et noeuds.
- Une contrainte de chemin basée sur la conservation du flot $\forall j, \sum_i x_{ij} = \sum_k x_{jk}$
- Une contrainte d'accessibilité (connexité) basée sur l'algorithme de Ford. Cette contrainte détermine si un noeud est accessible à partir de l'origine par un chemin de longueur k.

Heuristiques dédiées :

- Une heuristique de routage par plus court chemins. On suppose que toutes les demandes jusqu'à $k-1$ ont été routées (le réseau comporte donc des arcs déjà utilisés) et on considère pour la k-ième demande le routage qui minimise le coût total du réseau sans s'occuper des demandes suivantes. La stratégie de branchement fixe les arcs empruntés par k en suivant le chemin recommandé (le premier arc est imposé ou interdit récursivement).
- Une procédure d'accélération de l'heuristique précédente par DBDFS [1]. Lorsque l'on dispose d'une bonne heuristique il est souvent intéressant de modifier l'ordre de visite des noeuds afin de privilégier la découverte rapide de bonnes solutions.

Corrections adhoc de défauts de la méthode mis en évidence par les jeux de test :

- Les tests ont montré qu'il était plus intéressant de considérer les demandes à router non en ordre décroissant de volume mais suivant le paramètre $2 \times \mathsf{Vol}_{ij} + \mathsf{Vol}_{ji}$ où Vol_{ij} est le volume des demandes d'origine i et destination j.
- Afin de revenir rapidement sur les premières décisions DBDFS a été remplacée par une procédure qui réordonne les noeuds à évaluer en fonction d'une pénalité variable appliquée aux branches droites selon que le noeud courant a une profondeur inférieure à 60 ou supérieure à 240.
- L'ouverture de nouveaux arcs a été pénalisée (multiplication du coût réel des liens par 2) afin de diriger le solveur vers des réseaux de basse densité.
- Une procédure de post-optimisation par recherche locale a été mise en place avec pour voisinage la possibilité d'éliminer tout arc de la solution courante. Le voisinage est exploré par recherche arborescente avec le solveur précédent.

Perron [13] [14] [15] [6] encapsule le solveur précédent dans une approche à grands voisinages (LNS). Les caractéristiques de cette approche sont :

- Des voisinages structurés. Deux arcs sont choisis au hasard et toutes les demandes passant pas ces arcs sont reroutées.
- Une approche portfolio adaptative. Selon le type d'instance, différents paramètres de la recherche (pénalisation des branches droites, modifications de la fonction coût, limite de retour en arrière) se

sont montrés efficaces. Un algorithme adaptatif assure un apprentissage des paramètres afin de s'adapter au mieux à l'instance considérée.
- Réinitialisation de la recherche (*fast restart*). Comme le solveur tend à plonger dans une branche et à ne jamais pouvoir prouver qu'elle ne conduira pas à une meilleure solution, on le force à remonter à la racine lorsqu'une certaine limite de retours en arrière (*fail limit*) est franchie.
- Diversification par randomisation. L'ordre dans lequel les demandes sont considérées pour le routage est randomisé afin d'amortir l'impact des erreurs commises par le solveur lors de ses premières décisions.

Un effort semblable a été effectué en programmation linéaire et les différents algorithmes ont été parallélisés.

Les résultats complets sont disponibles sur la toile à l'adresse `wwwex.prism.uvsq.fr/Rococo`.

3 Evolution du problème

3.1 Instances C de Rococo

Le problème Rococo est une simplification d'un problème de dimensionnement de réseaux beaucoup plus complexe si bien que dès sa définition, la nécessité d'évolution a été prise en considération. Ainsi le jeu d'instances C est formé d'extensions du problème :
- Problème à liens non-symétriques (C11)
- Problème d'extension de réseaux déjà construits (C16)
- Problèmes avec topologie fixée (C10, C16)
- Problème à demandes hétérogènes avec qualités de service différentes (C20)

Ces extensions sont cependant limitées par la conception même de Rococo.

Un exemple en est l'extension aux problèmes à liens non-symétriques. Dans Rococo, le graphe support est non-orienté si bien que les liens se présentent sous forme de couples ($\mathsf{capacite}_{ij}$, $\mathsf{capacite}_{ji}$) par exemple $(0, 0)$, $(64, 64)$. Pour simuler un problème orienté, l'instance C11 ajoute des liens asymétriques $(0, 64)$ et $(64, 0)$.

Un autre exemple est le problème C16 dans lequel la topologie du graphe est contrainte à prendre la forme d'une chaîne ou d'une boucle en jouant sur les paramètres de multiplicité ($m_{ij} \leq 1$) et de ports ($\mathsf{PortsMax}_i \leq 2$).

3.2 Extensions possibles

Lors de la conception de Rococo, les extensions suivantes avaient été envisagées pour être finalement écartées :

Protection 1 + 1 : Pour tout demande il y a une route arc-disjointe de protection pour prévenir d'éventuelles pannes du réseau.

Panachage : Plusieurs liaisons de capacités égales ou non sont autorisées entre deux noeuds i et j.

Différentiation des noeuds : On peut interdire d'utiliser certains noeuds comme noeuds de transit. Ces noeuds ne peuvent alors être que des sources ou des puits de trafic.

Routage uniquement selon la destination : Pour des raisons liées à l'implantation de tables de routage, le concepteur du réseau peut imposer qu'en sortie d'un noeud i donné, l'ensemble du trafic destiné à un noeud j donné emprunte le même arc.

Topologies particulières : Le concepteur peut souhaiter se limiter à des topologies de réseau particulières par exemple un arbre, un cycle ou de cycles reliés par une unique arête.

De manière plus générale, les contraintes supplémentaires envisageables se classent en trois catégories :
- contraintes topologiques : s'exprimant sur les variables y_{ij} uniquement, elles imposent une certaine topologie au réseau résultant, par exemple une limite sur le degré des noeuds du réseau ou encore être un arbre.
- contraintes sur les routages : s'exprimant sur les variables x_{ij}^d uniquement, elles imposent des conditions aux routes par lesquelles circulent les demandes, par exemple une longueur maximale, un flot maximal en certains noeuds, etc.
- contraintes couplantes : elles portent à la fois sur les variables y_{ij} et sur les variables x_{ij}^d, par exemple certaines demandes dites "sécurisées" ne peuvent circuler que dans des arcs munis de liens sécurisés également.

3.3 Le problème Fado

Il a été choisi de faire évoluer le problème en donnant la priorité aux contraintes topologiques sous-représentées dans Rococo, tout en laissant une grande liberté au concepteur du réseau dans le domaine des contraintes sur les routages.

Le problème Fado est donc une évolution de Rococo dans laquelle :
- La contrainte de sécurité est éliminée.
- Une contrainte optionnelle d'organisation du réseau sous forme de réseau dorsal de haut débit relié à un réseau capillaire de bas débit.
- Toute contrainte de routage doit pouvoir être utilisée, autrement dit les solveurs ne peuvent pas

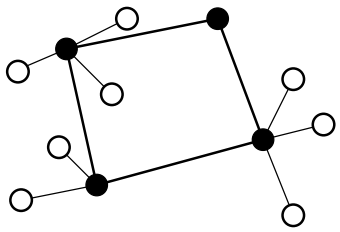

FIG. 1 – Réseau formé d'une dorsale de haut débit (concentrateurs) et d'un réseau capillaire (noeuds terminaux reliés par des liens de bas débit).

s'appuyer sur une structure particulière des routages autre que le routage par des chemins (par opposition au routage par flots).

Désormais un noeud peut être terminal ou concentrateur (c'est à dire routeur). Un noeud terminal est un noeud qui ne peut servir d'intermédiaire si bien que toutes les demandes qui lui parviennent doivent lui être destinées. Un noeud terminal doit être directement relié a un unique concentrateur et comme le trafic y est limité cela peut se faire par un lien de faible qualité dit "bas débit". Les concentrateurs en revanche sont liés entre eux par des liens "haut débit". Le type d'un noeud peut être imposé (ou pas). La figure 1 montre un exemple de réseau hierarchisé.

D'ordinaire la résolution de ce type de problème se fait en 2 phases. Dans un premier temps on décide quels noeuds seront terminaux et à quel noeuds ils seront rattachés, puis on dimensionne le réseau classiquement. Le problème de localisation de concentrateurs est un problème de la famille des problèmes de rangement (*packing*) et de localisation (*facility location*) le plus souvent résolu par PLNE. Gourdin, Labbé et Yaman font un tour d'horizon des variantes de ce problème et méthodes de résolution, pour le cas spécifique des problèmes de télécommunications [8].

Pour modéliser cette nouvelle dimension du problème, on introduit les variables t_i qui indiquent si le noeud i est un noeud terminal et les variables bd_{ij} qui indiquent si l'on peut installer un lien de bas débit sur l'arc (i,j) et l_{ij} qui indiquent si sur l'arc (i,j) a été installé un lien non nul. La contrainte de hiérarchie du réseau s'exprime par la conjonction des 3 contraintes suivantes : Si un noeud est terminal, il ne peut être que relié à des noeuds concentrateurs $[\forall i,\ t_i = 1] \Rightarrow [\forall j,\ l_{ij} \neq 0 \Rightarrow t_j = 0]$. Si un noeud est terminal son degré est 1 $[\forall i,\ t_i = 1] \Rightarrow [\sum_j l_{ij} = 1]$. Enfin, un lien de bas débit doit avoir exactement une extrémité qui est un concentrateur et l'autre qui est un terminal $[bd_{ij} = 1] \Rightarrow [t_i + t_j = 1]$

Les nouvelles équations s'écrivent :

$$\forall ij,\ l_{ij} \ = \ [y_{ij} \neq 0] \tag{14}$$

$$\forall i,\ t_i \ + \ \left(\sum_j l_{ij} t_j \right) \ \leq \ 1 \tag{15}$$

$$\forall i,\ [t_i = 1] \Rightarrow \left[\sum_j l_{ij} = 1 \right] \tag{16}$$

$$\begin{cases} \forall ij,\ bd_{ij} + t_i + t_j \ \leq \ 2 \\ \forall ij,\ bd_{ij} \ \leq \ t_i + t_j \end{cases} \tag{17}$$

$$t_i \in \{0,1\} \quad bd_{ij} \in \{0,1\} \quad l_{ij} \in \{0,1\}$$

4 Des moyens "élémentaires" pour résoudre un problème complexe

Notre approche est basée sur l'utilisation de la modélisation au niveau atomique du solveur, la consistance forte et la recherche par défaut. Nous expliquons dans cette section chacun de ces termes.

4.1 Modélisation au niveau atomique

Lors du projet Rococo il avait été nécessaire d'écrire plusieurs contraintes dédiées qui sont pourtant très proches des contraintes déjà disponibles dans le solveur, par exemple le produit scalaire ou la contrainte de conservation du flot.

La difficulté vient de l'utilisation d'une structure de données dédiée – les variables de graphe – qui est opaque pour le reste du solveur. Dans le problème Fado, de nouvelles contraintes doivent pouvoir être aisément ajoutées donc une telle approche est exclue. Nous avons donc choisi de ne pas transformer le modèle en gardant le modèle à variables booléennes.

C'est une approche tout à fait contraire à ce qui est recommandé par exemple par Smith dans le chapitre consacré à la modélisation du Handbook of constraint programming [17].

4.2 La consistance forte

La consistance forte pour une contrainte est la simulation d'un algorithme de propagation "parfait" avec un second solveur de contraintes. On la rencontre sous les 3 formes, dynamique, semi-dynamique et statique.

Dynamique : un second solveur vérifie en tout point de l'arbre de recherche s'il existe une solution de la contrainte contenant l'affectation $x = v$. Dans le cas contraire, la valeur v est éliminée du domaine de x. Cette approche semble avoir été introduite par Régin [3].

Statique : au moyen d'un second solveur on génère toutes les solutions d'une contrainte que l'on mémorise dans une table. On ajoute ensuite au modèle une contrainte spécifiant que seules les affectations contenues dans la table sont autorisées. Plusieurs solveurs de contraintes disposent de moyens plus ou moins équivalents pour la mettre en oeuvre [17].

Semi-dynamique : les solutions sont générées à la volée comme dans le cas dynamique mais sont sauvegardées comme dans le cas statique dans une table dite "ouverte" (car la contrainte correspondante ne fait pas de négation par l'absence). On évite ainsi que les mêmes solutions ne soient générées plusieurs fois dans des noeuds successifs.

La consistance forte permet donc de détecter si la propagation "parfaite" d'une contrainte donnée permet de résoudre plus efficacement un problème. Si c'est le cas, nous avons identifié un sous-problème critique sur lequel il convient de se pencher.

Même si ILOG CP Optimizer permet les trois approches, nous nous limiterons ici à l'approche statique.

4.3 La recherche par défaut

ILOG CP Optimizer comporte désormais une stratégie de recherche prédéfinie basée sur la propagation [16]. La paramétrisation la plus simple consiste à grouper les variables en *phases de recherche*. Le solveur devra avoir fixé toutes les variables de la phase k avant de brancher sur celles de la phase $k+1$. Tous les autres choix sont laissés aux heuristiques internes du solveur.

Il existe des façons plus directives pour paramétrer la recherche prédéfinie mais nous n'en auront pas usage.

5 Résoudre simplement le problème de dimensionnement de réseaux

Nous allons d'abord montrer qu'une approche basée sur les seuls chemins ne saurait résoudre le problème, les bonnes performances de CP-LNS (basée sur des chemins) s'expliquant par un effet de bord de la stratégie LNS. Nous montrerons ensuite qu'une approche purement topologique ne saurait convenir, même avec une consistance forte sur les chemins. Une approche intermédiaire a donc été finalement retenue.

5.1 Approches orientées chemins

La mise en place du problème de base est une transcription presque à l'identique du modèle mathématique $(4) - (13)$.

Grâce à la recherche prédéfinie, on peut de suite en évaluer les performances. Le temps est limité à 600 secondes et il s'agit des problèmes 011000 (pas de multiplication et routage symétrique). La plateforme de test est un Pentium III 700 (plateforme normalisée du projet Rococo).

problème	solution	temps	fails	déviation
A04	22267	3	3850	0 %
A05	30744	57	30817	0 %
A06	37716	303	108047	0 %
A07	48260	600	112706	1 %
A08	66437	600	77457	17 %
A09	90067	600	46647	27 %
A10	102226	600	88455	40 %

A titre de comparaison voici les premiers résultats obtenus lors du projet Rococo [6]. MIP désigne un modèle simple résolu avec Cplex, CG une approche par génération de colonnes, CP-PATH le solveur décrit précédemment sans grands voisinages.

instance	optimum	MIP	CG	CP-PATH
A07	44728	44728	44728	47728
A08	56576	56576	57185	56576
A09	70885	73180	72133	70885
A10	82306	99438	87148	83446

Cependant, les résultats s'effondrent sur les instances B avec des déviations à la meilleure solution connue supérieures à 300%.

5.1.1 Stratégies de recherche orientées chemin

Nous désirons évaluer dans quelle mesure il est intéressant de s'investir dans des stratégies de recherches basées sur la notion de plus court chemin telles celles développées par ILOG.

On peut se faire une idée du gain apporté par une telle stratégie en paramétrant la recherche par défaut. Il suffit de déclarer une phase de recherche pour chaque demande, prises en ordre décroissant de taille, puis une phase pour les autres variables du problème. Le solveur va donc instancier entièrement le chemin de la première demande, puis la seconde, et ainsi de suite, en appliquant à l'intérieur de chaque phase ses propres heuristiques.

Cette modification réduit considérablement le temps de découverte de solutions de qualité acceptable mais n'améliore que faiblement la preuve d'optimalité. De plus, la recherche prédéfinie se montre globalement meilleure quand la taille des problèmes augmente. Les résultats restent médiocres sur les instances B.

problème	solution	temps	fails	déviation
A04	22267	1	1422	0 %
A05	30744	33	20432	0 %
A06	37716	176	59992	0 %
A07	49039	600	17299	3 %
A08	64011	600	81148	13 %
A09	89498	600	50615	26 %
A10	116084	600	27416	41 %

Nous avions à disposition une stratégie de recherche écrite précédemment, K-PATH, basée sur l'énumération des k plus courts au moyen de l'algorithme d'Eppstein [7] en prenant tout particulièrement soin de réduire les calculs lors du retour en arrière avec des algorithmes de réoptimisation de type primal-dual [11]. K-PATH est très efficace sur les instances A, la solution optimale de A08 est obtenue en 6 secondes et 10291 retours en arrière, sans preuve. Mais elle s'avère décevante sur les instances B ou C. Sur l'instance B15 elle découvre une solution de 54616 en 5 secondes, 54277 en 70 secondes puis reste bloquée pendant 10 minutes alors que des solutions de valeur 26000 sont connues. Il en est de même pour CP-PATH dont la première solution pour B15 est 45927.

Les résultats de la recherche par défaut ont donc été un bon indicateur des (mauvaises) performances auxquelles il aurait fallu s'attendre pour une stratégie orientée chemin, même très perfectionnée telle K-PATH ou CP-PATH.

5.1.2 Contraintes de chemin

Nous désirons à présent évaluer l'intérêt de s'investir dans la réalisation d'une contrainte de chemin spécialisée comme celle développée par Bourreau et Cambazard [5]

A cette fin, nous allons simuler une contrainte "parfaite" au moyen des tables d'ILOG CP Optimizer. Il suffit de demander au solveur de générer toutes les affectations possibles des variables x_{ij}^d pour d fixé et $(i,j) \in G$ et de les sauvegarder dans une table, puis de poser la contrainte de table correspondante.

Cela n'est possible que pour les petites instances dans le cas général, mais si la contrainte limitant la longueur des chemins est active, on peut tester toutes les instances de la série A (ainsi, A10 000100 comporte 62 tables de 65 chemins avec 100 entiers par chemin). Pour des problèmes plus grands, on peut limiter artificiellement la longueur des chemins.

Notre test consiste à essayer d'obtenir la preuve d'optimalité des instance A avec les options 011100 (pas de multiplication, routages symétriques et limitation de longueur des chemins).

Mais la contrainte "parfaite" de chemins ne suffit pas même à en trouver les valeurs des solution optimales. Ainsi pour A07 la recherche prédéfinie trouve une solution de 49228 (62820 fails) et l'heuristique simple de chemins 48818 (106654 fails) sans preuve d'optimalité. Les différentes variantes donnent des résultats proches. Par exemple on peut ajouter dans la table aux $n(n-1)$ arcs d'un chemin les $n-1$ noeuds et sa longueur. Les résultats deviennent alors 49093 (58333) et 48818 (101196) respectivement. Utiliser une heuristique plus sophistiquée telle K-PATH permet d'obtenir certaines solutions optimales mais les preuves font toujours défaut.

Ainsi, même une contrainte de chemins "parfaite" ne saurait suffire à résoudre de façon satisfaisante les instances A. Les résultats sur les instances B sont tout aussi décevants.

5.2 Approches basées sur la topologie

Les résultats précédents nous conduisent à nous interroger sur la façon dont CP-LNS parvient à résoudre les instances B étant donné qu'elle est basée sur CP-PATH : un examen d'une exécution de CP-LNS sur B15 montre que la première solution trouvée est naturellement semblable à celle de CP-PATH, comptant 55 liens pour un coût de 45927. Ensuite deux liens de la solution sont choisis et les demandes passant par ces liens reroutées, mais la plupart du temps elles le sont sans ouverture de lien supplémentaire. Si bien que la solution suivante compte 53 liens pour un coût un peu amélioré. Et ainsi de suite jusqu'à 28460 pour 15 liens. CP-LNS produit donc des solutions de faible densité par un effet de bord de la recherche à grand voisinages.

Une idée simple à tester est une heuristique basée sur la topologie du réseau. On paramètre la recherche prédéfinie avec les phases de recherche en demandant au solveur de fixer dans l'ordre les groupes de variables suivants :

- phase topologique l_{ij}
- pour chaque d une phase de routage x_{ij}^d et x_i^d
- phase de choix du type y_{ij} et multiplicité m_{ij}

Cette simple heuristique donne des résultats meilleurs que les heuristiques complexes de chemins sur les instances de la série B. Cependant, quand la stratégie de branchement travaille sur la topologie, elle ne prend qu'indirectement en compte les contraintes sur les routages. Elle peut donc très aisément générer des structures topologiques qui n'admettent aucun routage réalisable.

Les contraintes dont il faut se soucier tout particulièrement sont :

- La limitation de longueur. Même si on oublie la composante capacitaire en considérant pour chaque lien une capacité nulle ou une capacité infinie de coût c_{ij} le problème de construction

d'un réseau de coût minimal sous contrainte de distances minimales entre certains de ses noeuds (*diameter constrained spanning tree*) est un problème difficile dans la pratique. D'ordinaire les méthodes de résolution introduisent justement des demandes fictives x^d et se ramènent à un problème de réseaux
- La limitation de degrés (pondérés si multiplicité). Le cas particulier à capacité infinie est connu sous le nom d'arbre couvrant à degrés contraints (*degree constrained spanning tree*). Il a également fait l'objet de nombreuses études et algorithmes d'approximation.
- La limitation de trafic. A notre connaissance il n'existe pas vraiment de problème répertorié qui soient proche de cette variante.

Dans tous les cas, l'aspect capacitaire complexifie notablement ces problèmes.

5.2.1 Une contrainte de distance

On déclare un tableau de variables d_{ij} dont la valeur est la distance minimale en nombre d'arcs entre chaque couple de noeuds du réseau. Ainsi, la limitation de longueur des chemins se transpose en une limitation de distance sur le réseau par l'intermédiaire de la contrainte

$$\forall d,\ d_{ij} \ \leq\ L^d \quad \text{avec } i = s^d, j = t^d$$

où la longueur L^d de la demande d peut être calculée par la somme des arcs $\sum_{ij} x_{ij}^d$ et la somme des noeuds $\sum_i x_i^d - 1$ (nous utilisons les deux simultanément).

On voudrait définir d_{ij} par rapport à l_{ij} et d_{kj} comme dans la relation récursive de Bellman

$$\forall ij,\ d_{ij} = \min_{\{k|l_{ik}=1\}} d_{kj} + 1$$

ILOG CP Optimizer fournit un opérateur `IloMin` et pour simuler l'opérateur de sélection $\{k \mid l_{ik} = 1\}$ nous allons considérer un majorant arbitraire M de toutes les distances et écrire une expression d_{kj}^{cond} telle que :
- si $l_{ik} = 0$ alors $d_{kj}^{\text{cond}} \geq M$
- si $l_{ik} = 1$ alors $d_{kj}^{\text{cond}} = d_{kj}$

Alors quand on prendra le minimum sur k des expressions d_{kj}^{cond}, toutes celles pour lesquelles $l_{ik} = 0$ deviendront plus grandes que M et n'influeront pas sur le résultat.

Il suffit de prendre $d_{kj}^{\text{cond}} = l_{ik}(d_{kj} - M) + M$ où M est le nombre de noeuds du graphe. On pose enfin

$$\forall ij,\ d_{ij} = 1 + \min_k \left[l_{ik}(d_{kj} - M) + M\right]$$

Attention aux cas $k = i$, $k = j$ et $i = j$.

Cette contrainte est équivalente à l'algorithme de Bellman, elle va donc calculer la distance entre tout couple de points (i, j) par point fixe en n^4 dans le pire des cas. Elle permet à l'heuristique topologique de générer peu de graphes infaisables.

5.2.2 Trafic et degrés : partition en étoiles

Les contraintes de trafic et de degrés s'appliquent sur les noeuds mais ont des conséquences globales sur le réseau. Par exemple un réseau dont le degré de tout noeud est 2 est nécessairement une boucle.

Une façon simple de les traiter est de lier leur formulation locale à un noeud et leur formulation globale au graphe. Pour ce faire, on partitionne les arcs du graphe en autant d'étoiles qu'il y a de noeuds et on exprime les grandeurs voulues en chaque noeud.

$$\text{capacite entrante}_i = \sum_j \mathsf{Capacite}_{ji}[y_{ji}] \tag{18}$$

$$\text{flot entrant}_i = \sum_j \sum_d \mathsf{Vol}^{\mathsf{d}} x_{ji} \tag{19}$$

$$\forall i,\ \text{flot entrant}_i \leq \text{capacite entrante}_i \tag{20}$$

$$\text{degre}_i = \sum_j y_{ij} \tag{21}$$

Ces grandeurs peuvent être reliées à leur contrepartie globale, par exemple l'expression du nombre de liens par rapport à la somme des degrés des noeuds ou le flot total par rapport à la longueur des chemins.

$$2 \times \text{liens} = \sum_i \text{degre}_i \tag{22}$$

$$\text{flot} + \sum_d \mathsf{Vol}^d = \sum_i \text{trafic}_i \tag{23}$$

$$\text{flot} = \sum_d \mathsf{Vol}^d \times L^d \tag{24}$$

$$\text{flot} = \sum_i \text{flot entrant}_i \tag{25}$$

On a ainsi relié des grandeurs globales comme la longueur d'un chemin ou le nombre de liens à des grandeurs locales comme le flot, le trafic ou la capacité en un noeud.

Malheureusement, si la décomposition en noeuds du problème améliore la prise en compte des contraintes de limitation de trafic et de degré lors d'une approche entièrement topologique, il y a de nombreuses instances de la série B où l'approche orientée topologie ne trouve aucune solution avec ces deux contraintes actives car la topologie générée est trop contrainte mais le solveur ne le voit que très tardivement lors de la phase de routage.

5.3 Approche intermédiaire

L'inconvénient des deux approches envisagées est qu'elles se situent chacune à un extrême du problème. L'approche orientée chemin ignore totalement la topologie du graphe, se contentant d'ouvrir des liens quand nécessaire sans vision globale du coût. L'approche orientée topologie ignore totalement les contraintes de routage.

On peut bien entendu "corriger" les défauts de chaque approche. Ce fut le cas dans les travaux précédents d'ILOG qui pénalisaient l'ouverture d'arcs dans les heuristiques orientées chemin (CP-PATH). Mais il ne saurait s'agir d'une solution satisfaisante sur le long terme.

Nous avons plutôt opté pour fixer dans un premier temps le fait qu'un noeud est terminal ou routeur. On peut ainsi fixer le type de tous les noeuds du graphe sans figer pour autant sa topologie mais en restreignant quand même les routes possibles pour les différentes demandes. C'est également le choix qui est fait implicitement quand le problème est décomposé en une phase de localisation des concentrateurs suivie d'une phase de dimensionnement classique [8].

Il suffit de paramétrer la recherche par défaut avec les phases suivantes :

- affectation des concentrateurs t_i
- pour chaque d une phase de routage x_{ij}^d et x_i^d
- phase de choix du type y_{ij} et multiplicité m_{ij}

Quelques contraintes supplémentaires sont utiles, par exemple remarquer qu'une demande entre deux noeud terminaux compte au moins 2 arcs.

Cette stratégie intermédiaire donne des résultats acceptables sur les deux jeux de données A et B. Par exemple sur B15 000000, la solution trouvée est 36406 la meilleure solution connue étant 26126, CP-LNS trouvant 28460 et Menne 33248.

Bien que nos solutions soient en moyenne 2 fois moins bonnes que les meilleures solutions connues, elles sont comparables à celles obtenues par exemple par Menne dans son travail de DEA [10] (nous obtenons de moins bonnes solutions que lui sur les petites instances et de meilleures sur les grandes) pour un effort bien moindre – ce dernier ayant développé des inégalités valides spécifiques à Rococo et des stratégies de recherche dédiées implantées au dessus de Cplex. Cambazard [4] fournit peu de résultats numériques dans son mémoire de DEA mais nous estimons qu'il doit en être de même.

Le gain le plus important est cependant la flexibilité en raison d'une approche plus générique basée sur la distinction entre contraintes de routages et contraintes topologiques et leur prise en compte équilibrée dans une stratégie de recherche reposant sur la recherche prédéfinie. C'est ainsi que les nouvelles contraintes de Fado s'intègrent sans difficulté et sans dégradation des performances.

6 Conclusion et directions de recherche

Nous avons revisité un problème complexe de dimensionnement de réseaux en n'utilisant que des méthodes élémentaires. Cet impératif de simplicité, la simulation de contraintes complexes par consistance forte et la facilité de paramétrisation de la stratégie de recherche d'ILOG CP Optimizer nous a permis de centrer nos efforts sur la découverte de sous-problèmes critiques afin de leur apporter une réponse adaptée.

Les performances de certains algorithmes développés précédemment ne sont certes pas égalées mais l'approche est encore perfectible et ouvre de nouvelles directions de recherche dont voici un aperçu :

6.1 Contraintes de sac-à-dos

Une amélioration évidente est de mieux relier la topologie du graphe au coût. La décomposition en étoiles contribue à cela mais de manière insuffisante. Par exemple, si l'on connaît le flot minimal entrant dans un noeud, on peut en déduire le coût minimal si le degré de ce noeud est 1, 2, ... et combiner ces coûts entre eux en exploitant les contraintes topologiques.

En chaque noeud le problème se ramène à un sac-à-dos sous contraintes de cardinalité :

$$\min \sum_k c_k x_k$$
$$\underline{S} \leq \sum_k a_k x_k \leq \overline{S}$$
$$\underline{C} \leq \sum_k x_k \leq \overline{C}$$

La combinaison des bornes inférieures ainsi obtenues est elle même un problème de type sac-à-dos sous contraintes.

6.2 Calcul incrémental des distances

Nous avons remplacé la contrainte de distance par un propagateur dédié basé sur l'algorithme cubique de Floyd-Warshall. Ce propagateur est déclenché à chaque fois qu'un lien est fermé donc la complexité de l'algorithme est *systématiquement* n^5. Nous n'exploitons aucunement les supports des plus courts chemins pour réduire en pratique cette complexité. A notre connaissance il n'y a pas de travaux en programmation par contraintes dans cette direction.

6.3 Isthmes et points d'articulation

Notre approche topologique du problème reste proche des contraintes de chemin (filtrage de la longueur, des arcs nécessaires ou impossibles) mais envisage ces questions du point de vue du graphe support (union des chemins utilisés), plutôt que des chemins individuellement.

Une étude de connexité parallèle à [5] mais du point de vue des graphes est sans doute intéressante. Par exemple, quand on interdit l'utilisation d'un arc du graphe support, on peut créer des noeuds obligatoires (points d'articulation) ou des arcs obligatoires (isthmes) pour certaines demandes. On en déduit des flots minimaux dans les arcs ou noeuds correspondants.

Remerciements

Cette étude a pour origine une question soulevée par Bourreau et Cambazard à JFPC 2006 portant sur l'utilité d'une contrainte "parfaite" de chemin combinée à une heuristique dirigée par le coût.

Références

[1] J. Christopher Beck and Laurent Perron. Discrepancy bounded depth first search. In *Second workshop on integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, 2000.

[2] Raphael Bernhard, Jacques Chambon, Claude Le Pape, Laurent Perron, and Jean-Charles Régin. Résolution d'un problème de conception de réseau avec parallel solver. In *Journées Francophones de Programmation Logique par Contraintes (JFPLC)*, pages 151–166, 2002.

[3] Christian Bessière and Jean-Charles Régin. Enforcing arc consistency on global constraints by solving subproblems on the fly. In *5th International conference on principles and practice of constraint programming (CP)*, pages 103–117, 1999.

[4] Hadrien Cambazard. Conception de réseaux et optimisation combinatoire : étude d'un problème de multiflot monorouté sous contraintes opérationnelles. Mémoire de DEA, Institut de Recherches en Informatique de Nantes, 2003.

[5] Hadrien Cambazard and Eric Bourreau. Conception d'une contrainte globale de chemin. In *10e Journées nationales sur la résolution pratique de problèmes NP-complets (JNPC'04)*, pages 107–121, 2004.

[6] Alain Chabrier, Emilie Danna, Claude Le Pape, and Laurent Perron. Solving a network design problem. *Annals of Operations Research*, 130 :217–239, 2004.

[7] David Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2) :652–673, 1998.

[8] Eric Gourdin, Martine Labbé, and Hande Yaman. *Facility Location : Applications and Theory*, chapter 9. Telecommunication and Location. Springer, 2002.

[9] Laurent Jeannin and Simon de Givry. Network planning with constraint programming. In *International Network Optimization Conference (INOC)*, Evry/Paris, France, October 27-29 2003.

[10] Ulrich Menne. LP approaches to survivable networks with single path routing. Diploma Theses, Technische-Universität Berlin. Supervisor : Prof. Dr. Martin Grötschel, Prof. Dr. Rolf Möhring, 2003.

[11] Stefano Pallottino and Maria Grazia Scutellà. A new algorithm for reoptimizing shortest paths when the arc costs change. *Networks*, 31 :149–160, 2003.

[12] Claude Le Pape, Laurent Perron, Jean-Charles Régin, and Paul Shaw. Robust and parallel solving of a network design problem. In *8th International conference on principles and practice of constraint programming (CP)*, pages 633–648, 2002.

[13] Laurent Perron. Parallel and random solving of a network design problem. In *Workshop on Random Search of AAAI Conference (WAAAI)*, 2002.

[14] Laurent Perron. Practical parallelism in constraint programming. In *4th International workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pages 261–275, 2002.

[15] Laurent Perron. Fast restart policies and large neighborhood search. In *5th International workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, 2003.

[16] Philippe Refalo. Impact-based search strategies for constraint programming. In *10th International conference on principles and practice of constraint programming (CP)*, pages 557–571, 2004.

[17] Barbara M. Smith. *Handbook of Constraint programming*, chapter Chapter 11. Modelling, pages 377–406. Elsevier, 2006.