Optimiser ... en découpant des polyèdres

Renaud Sirdey

Résumé : Dans de nombreux domaines, les ingénieurs se trouvent confrontés à des problèmes combinatoires intrinsèquement difficiles. Une approche géométrique conduit à des méthodes, les *méthodes polyédriques*, qui permettent, en pratique, de les résoudre.

Mots-clés : Optimisation combinatoire - Polyèdres - Complexité.

Introduction

L'optimisation combinatoire est la branche des mathématiques qui traite des problèmes où il convient de trouver un meilleur élément parmi un ensemble de taille finie mais astronomique, ceci sans avoir recours à l'examen, impossible en pratique, de tous les éléments de l'ensemble.

Le problème bien connu du voyageur de commerce est emblématique. Il s'agit, étant donné un ensemble de villes et, pour chaque couple de villes, la distance qui les sépare, de trouver une tournée, c'est-à-dire un circuit passant une et une seule fois par chacune des villes, de longueur minimale. S'il y a n villes alors il y a $n!$ tournées possibles et, typiquement, 23! est du même ordre de grandeur que le nombre de microsecondes qui se sont écoulées depuis le *big bang* !

Un grand nombre de problèmes d'optimisation combinatoire sont NP-difficiles (c'est le cas de l'exemple ci-dessus), ce qui signifie qu'il est fort vraisemblable qu'il n'existe pas d'algorithme permettant de les résoudre qui soit efficace, autrement dit qui soit capable de faire significativement mieux, *dans le pire des cas*, que l'examen exhaustif du nombre exponentiel de solutions.

Ces résultats négatifs ne doivent pas obscurcir le tableau outre mesure : pour un problème NP-difficile donné, il existe généralement des familles d'instances particulières qui peuvent, elles, être résolues efficacement et, surtout, il peut s'avérer que la majeure partie des instances rencontrées en pratique ne soient pas si difficiles.

Une approche géométrique conduit à des méthodes, les *méthodes polyédriques*, qui sont actuellement parmi les plus performantes pour la résolution exacte de ces problèmes combinatoires difficiles.

1. Complexité des algorithmes

La complexité d'un algorithme est une fonction de la taille du problème (par exemple le nombre de villes dans le cas du problème du voyageur de commerce) qui fournit, à un facteur multiplicatif près, une borne supérieure sur le temps d'exécution de l'algorithme. Dans la mesure où cette borne est indépendante de l'instance, on parle de complexité au pire. On dit qu'un algorithme est *polynomial* ou tout simplement *efficace*, si cette fonction est un polynôme.

© L'OUVERT **115** (2007)

Identifier les algorithmes efficaces aux algorithmes polynomiaux est assez naturel. Par exemple, pour un problème linéaire (polynomial de degré 1) doubler la vitesse de l'ordinateur utilisé pour le résoudre revient *grosso modo* à doubler la taille des instances que l'on peut résoudre dans un intervalle de temps donné. Pour un problème dont la complexité est exponentielle, par contre, doubler la vitesse de l'ordinateur permet seulement d'ajouter une constante à la taille des instances que l'on peut traiter dans le même intervalle de temps !

Il existe de nombreux problèmes d'optimisation combinatoire qui peuvent être résolus efficacement, on dit aussi *en temps polynomial*. Par exemple, le problème du plus court chemin dans un graphe, le problème du flot de plus petit coût sur un réseau de transport et les problèmes de couplages sont tous des problèmes polynomiaux.

Néanmoins, il existe aussi de nombreux problèmes que l'on ne sait pas, aujourd'hui, résoudre efficacement, il s'agit des problèmes NP-difficiles.

L'ensemble NP est l'ensemble des problèmes de décisions, c'est-à-dire des questions dont la réponse est soit oui soit non, tels que, pour chacune des instances dont la réponse est oui, il existe un certificat qui permet de montrer en temps polynomial que la réponse est bien oui. Par exemple, si la question est de savoir, pour une instance du problème du voyageur de commerce, s'il existe une tournée de longueur au plus L alors, si la réponse est positive, il suffit d'exhiber une tournée qui vérifie cette propriété et il suffit alors de la parcourir, ce qui ne requiert qu'un temps proportionnel au nombre de villes.

L'ensemble des problèmes de décision que l'on peut résoudre en temps polynomial est noté P et il est généralement conjecturé que $P \neq NP$, autrement dit que certains problèmes de NP ne peuvent pas être résolus en temps polynomial. Cette conjecture fait partie des sept célèbres problèmes du millénaire dotés par la fondation Clay d'un prix d'un million de dollars[1].

Il y a bien sûr des problèmes de décision qui ne sont pas dans NP, certains problèmes de décision ne sont même pas décidables auquel cas il n'existe tout simplement pas d'algorithme permettant systématiquement d'obtenir une réponse en temps fini ! Le problème de l'arrêt d'une machine de Turing est emblématique de cette dernière famille.

Il existe dans NP des problèmes qui ont la particularité d'être polynomialement équivalents à tous les autres problèmes de NP, il s'agit des problèmes NP-complets. L'existence de tels problèmes est riche de conséquences : les problèmes NP-complets sont les plus difficiles de NP et s'il existe un algorithme polynomial qui permet de résoudre l'un des problèmes NP-complets alors cet algorithme peut être utilisé pour résoudre tous les problèmes de NP (NP-complets ou non) en temps polynomial.

Jusqu'ici nous avons uniquement considéré des problèmes de décision or nous sommes intéressés par des problèmes d'optimisation. Il est clair que si l'on sait résoudre un problème d'optimisation, par exemple, trouver une tournée de voyageur de commerce de longueur minimale alors on sait répondre à toutes les questions de la forme « existe-t-il une tournée de voyageur de commerce de longueur au plus L ? ». Inversement, si l'on sait répondre aux questions de la forme « existe-t-il une tournée de voyageur de commerce de longueur au plus L ? » alors on sait aussi résoudre le problème d'optimisation associé, par exemple, à l'aide d'une recherche dichotomique. Le nombre de questions à poser est alors de l'ordre du nombre de bits nécessaire pour coder les distances. Ceci permet donc de définir la notion

[1] Voir sur la toile à l'adresse `www.claymath.org/millennium/`.

de problème d'optimisation NP-difficile : un problème d'optimisation NP-difficile est un problème d'optimisation qui est au moins aussi difficile qu'un problème NP-complet.

2. Polyèdres

Un *polyèdre*[2] consiste en l'intersection d'un nombre fini de demi-espaces fermés, il s'agit donc d'un ensemble que l'on peut écrire sous la forme $\{x \in \mathbb{R}^n : Ax \leq b\}$ où A est une matrice $m \times n$ à valeurs réelles et où $b \in \mathbb{R}^m$. Chaque ligne de la matrice A définit, avec le coefficient du vecteur b associé, l'hyperplan frontière du demi-espace associé.

Un polyèdre est de *pleine dimension* si son intérieur (c'est-à-dire $\{x \in \mathbb{R}^n : Ax < b\}$) n'est pas vide.

Un hyperplan $\{x \in \mathbb{R}^n : \alpha^T x = \beta\}$ *supporte* un polyèdre s'il ne le coupe pas en deux et si son intersection avec le polyèdre n'est pas vide ; cette intersection définit alors une *face* du polyèdre. Les faces de dimension 0 et 1 s'appellent respectivement les *sommets* et les *arêtes* du polyèdre. Les faces de dimension maximale ($n-1$ si le polyèdre est un sous-ensemble de $\mathbb{R}^n$ de pleine dimension) s'appellent des *facettes*, ce sont les faces qui « collent » au mieux au polyèdre. La figure 1 illustre ces notions.

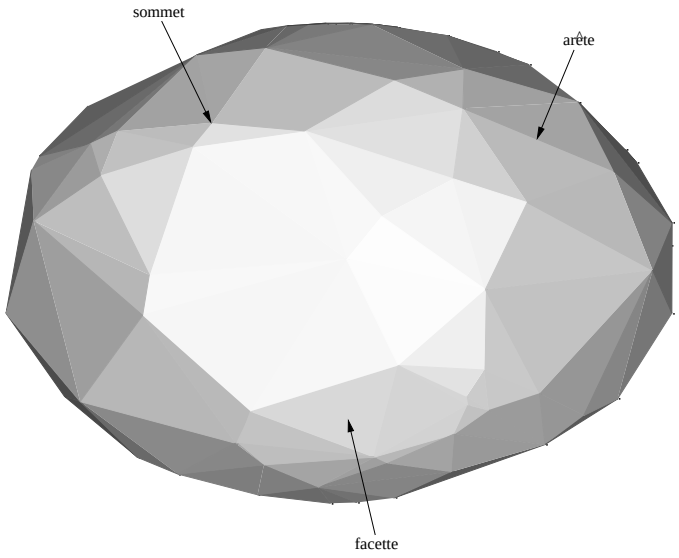

Fig. 1 – Exemple de polyèdre de pleine dimension (dans le cas contraire, il serait inclus dans un plan et, par conséquent, son intérieur serait vide) inclus dans $\mathbb{R}^3$. Les sommets et les arêtes sont des ensembles de dimension 0 et 1, respectivement. Ici, les facettes sont des ensembles de dimension 2. Les niveaux de gris ont été attribués aux facettes par simulation d'un éclairage de face.

Un polyèdre borné (soit contenu dans une boule de rayon fini) s'appelle un *polytope*. Tout polytope correspond à l'enveloppe convexe de ses sommets.

[2]Les polyèdres considérés ici sont toujours convexes.

3. La programmation linéaire

L'invention, en 1947, de la programmation linéaire, indissociable de celle de l'algorithme du simplexe, par G.-B. Dantzig est l'un des tours de force mathématiques du vingtième siècle.

Formellement, un programme linéaire consiste, étant donné un vecteur $c \in \mathbb{R}^n$, une matrice A, $m \times n$, à valeurs réelles et un vecteur $b \in \mathbb{R}^m$, à trouver un vecteur $x \in \mathbb{R}^n$, solution du programme mathématique suivant :

$$\begin{cases} \text{Minimiser } c^T x, \\ \text{sous la contrainte} \\ Ax \leq b. \end{cases}$$

Les premières applications de la programmation linéaire ont concerné les plans de formation, d'approvisionnement et de déploiement des troupes de l'armée de l'air américaine. Ces plans, appelés *programmes*, ont donné leur nom à la programmation linéaire puis, peu après, à la programmation mathématique en général.

L'exemple probablement le plus classique d'application de la programmation linéaire consiste à déterminer le régime alimentaire le moins cher qui soit conforme aux recommandations des nutritionnistes. Soit D un ensemble de denrées alimentaires et N un ensemble de nutriments (glucides, protides, lipides, etc.). On note c_d le coût unitaire de la denrée d et A_{nd} la quantité unitaire du nutriment n apportée par la denrée d. Enfin, b_n^- et b_n^+ dénotent respectivement les apports quotidiens minimum et maximum recommandés. Soit x_d la quantité de la denrée d à acheter, le problème s'énonce alors comme suit

$$\begin{cases} \text{Minimiser } \sum_{d \in D} c_d x_d, & \\ \text{sous les contraintes} & \\ b_n^- \leq \sum_{d \in D} A_{nd} x_d \leq b_n^+ & \forall n \in N, \\ x_d \geq 0 & \forall d \in D. \end{cases}$$

Contrairement aux apparences, un programme linéaire est un problème d'optimisation combinatoire. En effet, un tel programme possède une interprétation géométrique assez intuitive : dans la mesure où chaque ligne de la matrice A définit, avec le coefficient du vecteur b associé, un hyperplan, le problème revient à optimiser une forme linéaire sur un polyèdre, l'optimum (s'il existe) étant forcément réalisé en l'un de ses sommets. Puisque ces derniers sont en nombre fini, il s'agit bien d'un problème de nature combinatoire.

Généralement, on résout les programmes linéaires à l'aide de l'algorithme du simplexe. Grossièrement, l'algorithme du simplexe part d'un sommet du polyèdre[3] puis, à chaque itération, emprunte l'une de ses arêtes de manière à atteindre un *meilleur* sommet voisin. Cette dernière opération requiert la résolution de deux systèmes linéaires accompagnée de quelques manipulations. Ce schéma est répété jusqu'à ce que le sommet courant soit au moins aussi bon que tous ses voisins, donc optimal. La figure 2 illustre le fonctionnement de l'algorithme. Bien que son efficacité pratique soit remarquable, l'algorithme du simplexe n'est pas un algorithme polynomial : il existe des instances pathologiques à n variables et $2n$ contraintes qui requièrent un nombre exponentiel, $2^n - 1$, d'itérations.

[3]Trouver ce sommet nécessite la résolution, à l'aide de l'algorithme du simplexe, d'un programme linéaire dérivé du programme à résoudre pour lequel trouver un sommet initial est trivial.

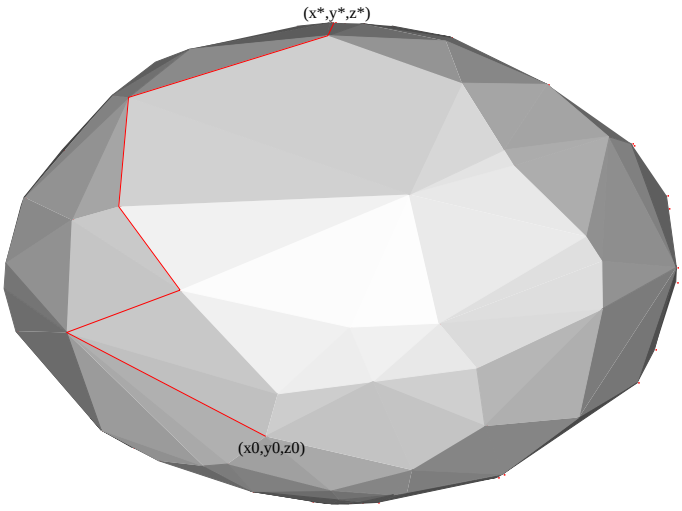

Fig. 2 – Exemple de chemin emprunté par l'algorithme du simplexe afin de trouver le sommet le plus élevé (soit maximiser y) du polyèdre ci-dessus. L'algorithme part du sommet (x_0, y_0, z_0) et chemine, en prenant systématiquement de l'altitude, jusqu'au sommet culminant (x^*, y^*, z^*).

Néanmoins, il existe des algorithmes polynomiaux capables de résoudre le problème de la programmation linéaire, en particulier l'algorithme de l'ellipsoïde introduit en 1979 par L. G. Khachiyan. Étant donné un polyèdre borné, donc un polytope, de pleine dimension (ces restrictions peuvent être levées au prix d'une technicité accrue) défini par un système linéaire d'inégalités, ce dernier algorithme fonctionne *grosso modo* comme suit. Au départ, on commence avec un ellipsoïde (la généralisation d'une ellipse à un nombre arbitraire de dimensions) suffisamment gros pour contenir toutes les solutions du système. À chaque itération, on vérifie si le centre de l'ellipsoïde correspond à une solution réalisable, si c'est le cas on s'arrête, sinon on choisit un hyperplan contenant le centre de l'ellipsoïde et qui le coupe de telle manière que l'une de ses moitiés contienne toutes les solutions réalisables. On choisit alors le plus petit ellipsoïde contenant cette dernière moitié et on itère jusqu'à ce que le volume de l'ellipsoïde soit en deçà d'un certain seuil, bien défini car un polytope (rationnel) de pleine dimension ne peut pas avoir un volume arbitrairement petit. La figure 3 illustre le fonctionnement de l'algorithme. Énoncé de cette manière, l'algorithme permet uniquement de décider si l'ensemble des solutions du système d'inégalités est vide ou pas. Une telle primitive, si tant est, ce qui est le cas ici, qu'elle soit polynomiale, permet néanmoins de résoudre le problème de la programmation linéaire en temps polynomial.

Bien qu'il ne puisse (paradoxalement ?) rivaliser, en pratique, avec l'algorithme du simplexe[4] et qu'il souffre de problèmes d'instabilité numérique, l'algorithme de l'ellipsoïde est, comme nous allons le voir, d'une importance théorique capitale.

[4]Contrairement à l'algorithme polynomial de point intérieur introduit en 1984 par N. Karmarkar.

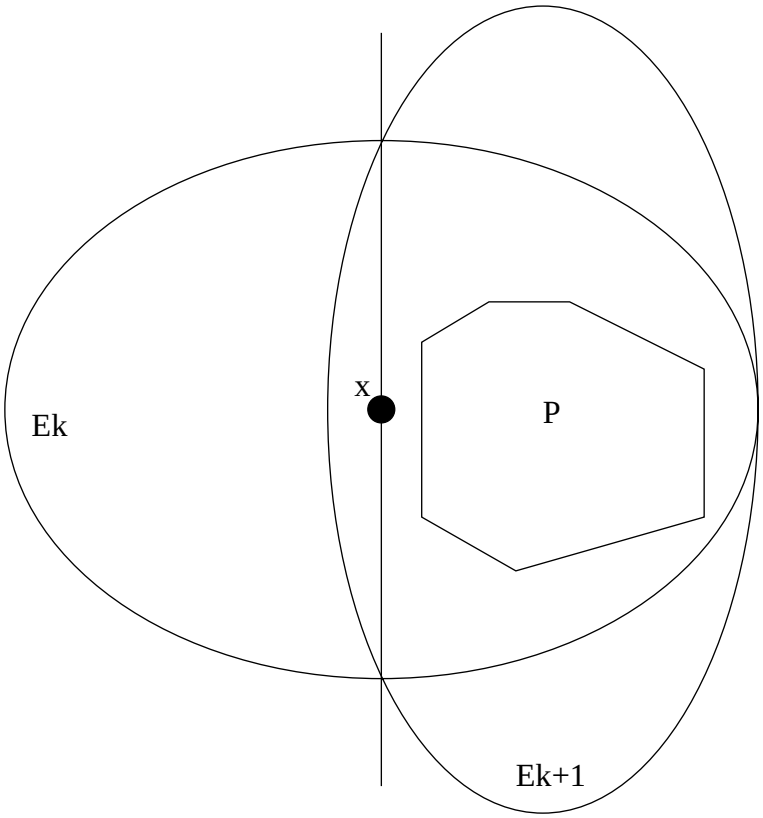

Fig. 3 – Illustration d'une itération de l'algorithme de l'ellipsoïde : l'ellipsoïde E_k contient le polytope P et son centre, x, n'appartient pas à P, on choisit donc un plan, passant par x, qui sépare x de P. La moitié droite de E_k, qui contient P, est choisie et E_{k+1} correspond justement au plus petit ellipsoïde qui contient cette moitié.

4. L'équivalence séparation/optimisation

En 1981, M. Grötschel, L. Lovász et A. Schrijver ont montré comment modifier l'algorithme de l'ellipsoïde afin de résoudre, bien sûr toujours en temps polynomial, des programmes linéaires sans que le système d'inégalités ne soit explicitement donné. Leur algorithme ne requiert qu'un *oracle séparateur*, c'est-à-dire un algorithme qui, étant donné un point de l'espace euclidien, décide s'il appartient au polyèdre ou fournit un hyperplan qui le sépare de ce dernier, qui n'est interrogé qu'un nombre polynomial de fois.

Muni de ce résultat, on peut, au moins en théorie, résoudre en temps polynomial des programmes linéaires à un nombre exponentiel (du nombre de variables) de contraintes à condition que le problème de séparation soit polynomial.

Plus généralement, si l'on sait séparer en temps polynomial alors on sait optimiser en temps polynomial (la réciproque est aussi vraie mais nous n'entrerons pas, ici, dans les détails). Il s'agit d'un résultat central de l'informatique théorique par le biais duquel on a pu montrer que des problèmes étaient polynomiaux bien avant qu'un algorithme « combinatoire » (c'est-à-dire exploitant directement la structure du problème) ne soit connu (on pourrait presque parler de preuves non constructives). Par ailleurs, certains problèmes polynomiaux attendent toujours leur algorithme « combinatoire »...

5. Résolution exacte des problèmes *NP*-difficiles

Ici, nous nous intéressons aux problèmes NP-difficiles que l'on peut écrire sous la forme d'un programme linéaire en nombres entiers, c'est-à-dire d'un programme mathématique de la forme

$$\begin{cases} \text{Minimiser } c^T x, \\ \text{sous les contraintes} \\ rAx \leq b, \\ x \in \mathbb{Z}^n. \qquad (I) \end{cases}$$

C'est le cas pour de nombreux problèmes d'optimisation combinatoire, NP-difficiles ou non.

La contrainte d'intégrité (I) ne change pas la nature du problème : il s'agit toujours d'optimiser une forme linéaire sur un polyèdre, le *polyèdre entier*, qui correspond à l'enveloppe convexe des solutions entières du système $Ax \leq b$. En effet, il existe un système linéaire d'inégalités minimal $A'x \leq b'$, principalement composé de facettes du polyèdre entier, tel que le problème ci-dessus est équivalent au programme linéaire

$$\begin{cases} \text{Minimiser } c^T x, \\ \text{sous la contrainte} \\ A'x \leq b'. \end{cases}$$

Malheureusement, à moins que la contrainte d'intégrité ne soit redondante, ce qui n'est que rarement le cas, le système $A'x \leq b'$ possède un nombre astronomique de lignes. Dans le cas des problèmes polynomiaux, ce n'est pas si grave, au moins en théorie, grâce, nous l'avons vu, à l'équivalence optimisation/séparation. Pour les problèmes NP-difficiles, par contre, le système $A'x \leq b'$ reste inaccessible dans son intégralité. Ceci étant dit, une connaissance partielle de ce système s'avère, en pratique, extrêmement pertinente.

En effet, une approche actuellement parmi les plus performantes pour la résolution exacte des problèmes NP-difficiles consiste à exploiter, dans le cadre d'un schéma de recherche arborescente, le programme linéaire continu obtenu en ignorant la contrainte d'intégrité (I), itérativement enrichi de facettes du polyèdre entier.

6. L'exemple du problème du voyageur de commerce

Pour se fixer les idées, revenons à notre problème de voyageur de commerce. Une instance à n villes peut être spécifiée, dans le cas symétrique, à l'aide d'un vecteur c à $\frac{n(n-1)}{2}$ composantes, chacune d'entre elles étant associée à une arête du graphe non orienté complet dont les n nœuds sont les villes. De même, une tournée peut être représentée à l'aide de son *vecteur caractéristique*, c'est-à-dire d'un vecteur x à $\frac{n(n-1)}{2}$ composantes binaires qui est tel que $x_e = 1$ si et seulement si l'arête e appartient à la tournée. Le problème revient alors à trouver un vecteur $x^\star$ qui est le vecteur caractéristique d'une tournée et dont la longueur,

$$c^T x^\star = \sum_{e \in E} c_e x_e^\star,$$

est minimale.

Le polyèdre défini comme l'enveloppe convexe des vecteurs caractéristiques des tournées s'appelle le *polytope du voyageur de commerce*. Il s'agit bien d'un polytope car il est contenu dans l'hypercube unité à $\frac{n(n-1)}{2}$ dimensions.

Pour mettre le problème sous la forme d'un programme linéaire en nombres entiers on procède généralement comme suit. Tout d'abord on impose les contraintes $0 \leq x_e \leq 1$ pour chaque arête e. Ensuite, on requiert que chaque ville soit visitée une et une seule fois ce qui revient à imposer qu'exactement deux des variables associées aux arêtes incidentes à un nœud v aient pour valeur 1. Formellement,

$$\sum_{e:v\in e} x_e = 2, \tag{1}$$

pour tout nœud $v \in \{1,\ldots,n\}$. Ces deux familles de contraintes permettent les vecteurs caractéristiques d'unions d'un ensemble de sous-tournées indépendantes, ces dernières n'étant bien évidemment pas des solutions admissibles. Pour les éliminer, il convient d'imposer que la tournée entre et sorte de chaque sous ensemble strict non vide de villes, autrement dit qu'au moins deux des variables associées aux arêtes dont un et un seul des nœuds est dans $\emptyset \subset V \subset \{1,\ldots,n\}$ aient pour valeur 1. Formellement,

$$\sum_{e:|V\cap e|=1} x_e \geq 2, \tag{2}$$

pour tout $\emptyset \subset V \subset \{1,\ldots,n\}$. Bien que ces dernières contraintes soient en nombre exponentiel, il y en a $2^n - 2$, elles sont séparables en temps polynomial.

Associé à la contrainte d'intégrité, l'ensemble des contraintes ci-dessus suffit à définir un programme linéaire en nombres entiers dont les solutions optimales sont les solutions optimales de l'instance du problème du voyageur de commerce associée.

Depuis les premiers travaux de G.-B. Dantzig et de R. Fulkerson, dans les années cinquante, de nombreux mathématiciens se sont attelés à l'étude du polytope du voyageur de commerce. Il en résulte, en particulier, tout un bestiaire de classes d'inégalités connues pour définir des facettes de ce polytope. Les inégalités (2) ci-dessus en font partie.

Comment, alors, résout-on le problème du voyageur de commerce ?

L'idée consiste à commencer par résoudre le programme linéaire continu comprenant les inégalités $0 \leq x_e \leq 1$, pour chaque arête e, et les égalités (1). Ce programme définit une *relaxation linéaire* du problème. Si, par chance, on obtient un vecteur solution caractéristique d'une tournée, alors on a fini. Sinon on recherche une ou plusieurs, généralement plusieurs, facettes qui sont violées par le vecteur solution courant et on les ajoute à la relaxation, de manière à « couper » la solution courante (voir la figure 4), que l'on résout à nouveau. Ce schéma est répété soit jusqu'à obtention d'un vecteur caractéristique d'une tournée, donc optimale, soit jusqu'à ce que l'on ne soit pas en mesure de trouver une nouvelle facette violée par la solution courante, rappelons en effet que l'on ne dispose que d'une connaissance très partielle du polytope entier. Dans ce dernier cas, la valeur de la solution non réalisable courante fournit un minorant de la longueur d'une tournée optimale.

Lorsque l'on se retrouve coincé, on scinde le problème en deux sous-problèmes en choisissant de fixer une variable, par exemple, à 0 et en résolvant le problème sous cette contrainte puis en fixant cette même variable à 1 et en gardant la meilleure des deux solutions obtenues. Ce schéma est appliqué récursivement : les sous-problèmes sont, si nécessaire, scindés en deux sous-sous-problèmes et ainsi de suite. L'algorithme décrit alors une arborescence dont les nœuds correspondent aux sous-problèmes.

Afin de réduire, autant que faire se peut, la taille de l'arborescence, l'algorithme mémorise la valeur de la meilleure tournée déjà rencontrée. Dans le cas où, lors de la résolution d'un

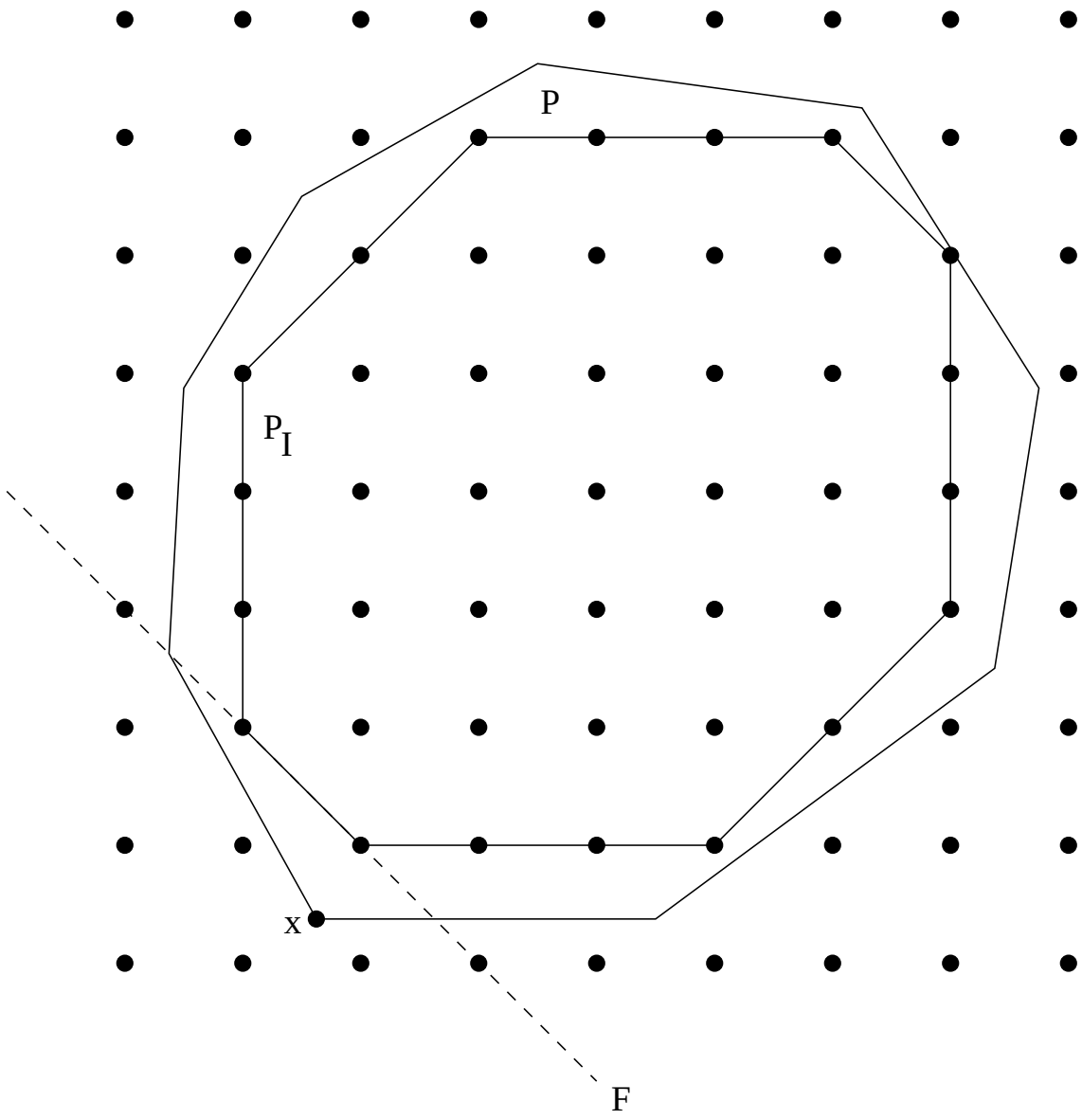

Fig. 4 – P est le polyèdre $\{x \in \mathbb{R}^2 : Ax \leq b\}$ et P_I est le polyèdre entier correspondant. La facette F sépare le sommet fractionnaire $x \in P$ de P_I.

sous-problème, l'ajout de facettes ne permet pas d'obtenir un vecteur caractéristique d'une tournée, il convient de s'assurer que le minorant obtenu n'est pas supérieur à la valeur de la meilleure solution déjà rencontrée. Si c'est le cas, résoudre le sous-problème est inutile. Ce principe permet de réduire le nombre de sous-problèmes à résoudre, donc le nombre de nœuds explorés, de manière drastique.

C'est à l'aide d'un algorithme de ce type, bien sûr beaucoup plus raffiné, et d'une bonne dose d'ingéniosité informatique que D. Applegate, R. Bixby, V. Chvátal et W. Cook ont réussi, en 1998, à résoudre une instance du problème du voyageur de commerce à plus de 13000 villes. Certes au prix de l'équivalent d'une dizaine de jours de calcul réparti sur un réseau d'une cinquantaine de stations de travail, ce qui ne diminue en rien la performance. La même équipe, renforcée par K. Helsgaun, détient le présent record : il est, depuis mai 2004, possible d'effectuer une tournée optimale des 24978 villes de Suède[5] !

[5] Voir sur la toile à l'adresse `www.tsp.gatech.edu`.

Malgré ces résultats empiriques remarquables, on sait aussi construire des instances de petite taille que l'on n'arrive pas, en pratique, à résoudre. Le problème du voyageur de commerce reste donc difficile, dans le pire des cas, et la conjecture $P \neq NP$ tient bon.

7. Quid des applications industrielles ?

Dans des domaines aussi variés que l'informatique, les télécommunications, le transport, la production, la finance ou la bioinformatique, les ingénieurs se trouvent régulièrement confrontés à des problèmes d'optimisation combinatoire, qu'ils soient NP-difficiles ou non. Par exemple, les problèmes d'affectation de fréquences dans les réseaux de téléphonie mobile ou les problèmes de conception de réseaux fiables, pour ne parler que du domaine des télécommunications, sont des problèmes NP-difficiles que l'on résout à l'aide des méthodes polyédriques.

Avec H. Kerivin du laboratoire d'informatique, de modélisation et d'optimisation des systèmes (Limos) de l'université de Clermont-Ferrand II, nous nous sommes attaqués à un problème de reconfiguration dynamique d'autocommutateurs répartis à l'aide, entre autres, des méthodes polyédriques. En quelques mots, il s'agit d'ordonner des déplacements de processus de traitement d'appels de telle manière que des contraintes de capacité sur les processeurs du système ne soient jamais violées, les situations de blocage étant résolues en arrêtant temporairement des processus, donc en renonçant temporairement à rendre une partie du service. Il s'agit alors de minimiser la dégradation temporaire du service induite par la reconfiguration du système.

Naturellement, notre démarche a été similaire à celle suivie pour le voyageur de commerce : après nous être assurés qu'il était NP-difficile, nous avons mis notre problème sous la forme d'un programme linéaire en nombres entiers, puis nous avons mis en évidence des classes de facettes du polytope entier associé et étudié les problèmes de séparation, enfin nous avons exploité ces résultats théoriques dans le cadre d'un algorithme de recherche arborescente.

Bien entendu, nos résultats ne sont pas aussi spectaculaires que ceux obtenus par les « tombeurs » du voyageur de commerce (soulignons néanmoins que des instances à plusieurs dizaines de milliers de déplacements ou de processeurs n'ont pas grand sens sur le plan de notre application industrielle). Typiquement, notre algorithme permet de résoudre en moins de quelques heures, la majeure partie des instances pratiques (de l'ordre, au maximum, de 80 déplacements pour des systèmes à quelques dizaines de processeurs) et, lorsque la taille des instances augmente et que l'on se retrouve contraint d'arrêter prématurément l'algorithme, le minorant obtenu par résolution de la première relaxation linéaire permet de prouver que la valeur de la meilleure solution rencontrée ne se trouve qu'à quelques pourcents de celle d'une solution optimale. Ce dernier point illustre une autre des qualités fondamentales des algorithmes de recherche arborescente polyédrique : ils fournissent généralement une bonne estimation au pire de l'écart à l'optimum lorsque le temps imparti ne suffit pas à résoudre le problème.

Remerciements

L'auteur souhaite remercier J. Carlier, professeur à l'université de technologie de Compiègne, et H. Kerivin, maître de conférences à l'université de Clermont-Ferrand II, qui ont bien voulu relire l'intégralité de ce texte.

Bibliographie

[1] D. Applegate, R. Bixby, V. Chvátal et W. Cook (1998), *On the solution of traveling salesman problems*, Documenta Mathematica **Extra Volume ICM**, 645–656.

[2] V. Chvátal (1983), Linear programming, *W. H. Freeman and Company.*

[3] G. B. Dantzig (1982), *Reminiscences about the origins of linear programming*, Operations Research Letters **1**, 43–48.

[4] K. Delvin (2002), The millenium problems, *Basic Books.*

[5] M. R. Garey et D. S. Johnson (1979), Computers and intractability, *W. H. Freeman and Company.*

[6] M. Grötschel, L. Lovász et A. Schrijver (1988), Geometric algorithms and combinatorial optimization, *Springer.*

[7] A. Schrijver (2004), Combinatorial optimization, *Springer.*

[8] R. Sirdey et H. Kerivin (2006), *A branch-and-cut algorithm for a resource-constrained scheduling problem*, RAIRO—Operations Research **à paraître**.

Renaud Sirdey
Nortel et UMR CNRS Heudiasyc
renauds@nortel.com