REVUE FRANÇAISE D'AUTOMATIQUE, INFORMATIQUE, RECHERCHE OPÉRATIONNELLE. RECHERCHE OPÉRATIONNELLE

J. ABADIE
H. DAYAN
J. AKOKA

Quelques expériences numériques sur la programmation non linéaire en nombres entiers

Revue française d'automatique, informatique, recherche opérationnelle. Recherche opérationnelle, tome 10, n° V3 (1976), p. 65-70

<http://www.numdam.org/item?id=RO_1976__10_3_65_0>

R.A.I.R.O. Recherche opérationnelle
(vol. 10, n° 10, octobre 1976, p. 65 à 70)

QUELQUES EXPÉRIENCES NUMÉRIQUES SUR LA PROGRAMMATION NON LINÉAIRE EN NOMBRES ENTIERS (*)

par J. ABADIE (1), H. DAYAN (2) et J. AKOKA (3)

Résumé. — *Il est rendu compte d'expériences numériques portant sur deux méthodes arborescentes, à l'aide de sept problèmes d'optimisation non linéaire totalement ou partiellement entiers.*

Notre objet est de rendre compte de quelques expériences numériques faites, à l'aide de méthodes de branchement, sur la résolution de programmes non linéaires en nombres entiers mixtes. Nous écrirons un tel programme, où $x \in \mathbf{R}^n$, sous la forme suivante :

$$\min \varphi(x), \tag{1}$$

$$f_i(x) \leqq 0, \qquad i \in I = \{1, \ldots, m\}, \tag{2}$$

$$a_j \leqq x_j \leqq b_j, \qquad j \in J = \{1, \ldots, n\}, \tag{3}$$

$$x_j \text{ entier}, \qquad j \in E \subset J. \tag{4}$$

Ce qui suit s'applique encore si l'on remplace dans (2), pour certains i ou même pour tous, le signe d'inégalité par le signe d'égalité, ou si la condition d'intégrité (4) est remplacée par la condition plus générale

$$x_j \in S_j, \qquad j \in E \subset J,$$

où S_j est un ensemble discret quelconque. Nous conserverons cependant la forme (1), (2), (3), (4) ci-dessus, qui enferme les problèmes-test utilisés plus loin.

Dans la suite, un problème du type (1), (2), (3), (4) sera appelé un *programme entier* [en abrégé PE, ou PLE si la fonction φ et les inéquations (2) sont linéaires]; lorsqu'il n'y a pas de condition d'intégrité (4), nous dirons qu'il s'agit d'un *programme ordinaire* (en abrégé PO, ou PLO s'il s'agit d'un programme linéaire ordinaire). Le mot *solution* désigne une solution du système (2), (3), tandis qu'une *solution entière* est une solution de (2), (3), (4). Une *solution optimale* est une solution du problème (1), (2), (3), tandis qu'une *solution optimale entière* est une solution du problème (1), (2), (3), (4).

(*) Reçu mai 1976.
(1) Électricité de France.
(2) Ministère de l'Industrie et de la Recherche.
(3) E.S.S.E.C.

LES MÉTHODES UTILISÉES

La plus usuelle des méthodes de branchement, en ce qui concerne la programmation *linéaire* entière mixte, est basée sur la méthode de Land et Doig [13], auxquelles se sont ajoutées des idées dues à Little, Murty, Sweeney et Karel [15], Dakin [9], Beale et Small [5], Driebeck [10], Roy [18], Tomlin [19]. Sous sa forme la plus simple, la procédure peut être décrite comme suit.

A une étape quelconque, soit $\hat{x}$ la meilleure solution entière connue. On pose $\hat{\varphi} = \varphi(\hat{x})$, et $\hat{\varphi} = +\infty$ si aucune solution entière n'est connue. On possède une liste de PLO. La première liste est constituée du seul problème (1), (2), (3) (suppression des conditions d'intégrité dans le problème donné).

L'étape générale se décompose en quatre pas successifs.

1. *Si la liste est vide, le programme linéaire entier donné est résolu : si* $\hat{\varphi} < +\infty$, *alors* $\hat{x}$ *est une solution optimale; si* $\hat{\varphi} = +\infty$, *alors le programme linéaire n'a pas de solution.*

2. *Prendre le dernier problème* PP *de la liste et le résoudre. Soit* x^* *une solution optimale,* $\varphi^* = \varphi(x^*)$ (*on pose* $\varphi^* = +\infty$ *dans le cas où* PP *n'a pas de solution*). *Si* $\varphi^* \geqq \hat{\varphi}$, *oter* PP *de la liste et aller en* 1 *pour une nouvelle étape.*

3. *Si tous les* x_j^*, $j \in E$, *sont entiers, alors* x^* *est une solution entière meilleure que* $\hat{x}$. *Poser* $\hat{x} = x^*$, $\hat{\varphi} = \varphi^*$, *ôter le problème* PP *de la liste, et aller en* 1 *pour une nouvelle étape.*

4. *Choisir un* x_β *non entier,* $\beta \in E$. *Ajouter à la liste, dans un ordre à choisir, deux nouveaux problèmes, qui ne diffèrent de* PP *que par de nouvelles bornes imposées à* x_β :

(*a*) *une borne supérieure égale à* $[x_\beta]$;

(*b*) *une borne inférieure égale à* $\langle x_\beta^* \rangle$;

où $[x_\beta^*]$ *désigne le plus grand entier inférieur à* x_β^*, *et où* $\langle x_\beta^* \rangle = [x_\beta^*] + 1$.

Le choix de la variable x_β au pas 4, et de l'ordre dans lequel les deux problèmes y sont créés, ont fait l'objet de nombreux travaux en ce qui concerne la programmation *linéaire* (voir, outre les auteurs déjà mentionnés, Le Faou [14]). Quels que soient ces choix, la même variable x_β peut être choisie à des étapes différentes, si bien que le nombre des problèmes de la liste ne peut pas être borné *a priori*. La méthode peut donc avoir quelque inconvénient de mémorisation lorsqu'on veut l'utiliser entièrement en unité centrale de la machine.

L'un des auteurs du présent article a proposé [1] deux méthodes où le nombre de mémoires nécessaires est connu *a priori*. Nous ne rappelons ici que le principe de celle qui a fait l'objet, avec la méthode précédente, de notre expérimentation numérique.

Les pas 1, 2, 3 sont les mêmes que ci-dessus, et le pas 4 est remplacé par le suivant :

4'. *Choisir un* x_β^* *non entier,* $\beta \in E$. *Considérer les trois problèmes suivants :*

(*a*) *le problème déduit de* PP *en fixant* x_β *à la valeur* $[x_\beta^*]$;

(*b*) *le problème déduit de* PP *en fixant* x_β *à la valeur* $\langle x_\beta^* \rangle$;

(*c*) *le problème* PP *lui-même ayant été obtenu en fixant une certaine variable* x_α *à une valeur entière* $\tilde{x}_\alpha$ *par excès ou par défaut, augmenter cet excès ou ce défaut en fixant, selon le cas, cette valeur à* $\tilde{x}_\alpha + 1$ *ou* $\tilde{x}_\alpha - 1$ [*ce problème* (*c*) *n'existe pas à la première étape*].

Si aucun des problèmes (*a*) *ou* (*b*) *n'a été résolu, les ajouter à la liste, dans un ordre à choisir, et aller en* 2 *pour une nouvelle étape; dans le cas où les problèmes* (*a*) *et* (*b*) *ont été tous deux résolus, supprimer* PP *de la liste, y ajouter* (*c*), *et aller en* 2; *dans le cas enfin où un et un seul des problèmes* (*a*), (*b*) *a été résolu, supprimer* PP *de la liste, y ajouter* (*dans l'ordre*) *le problème* (*c*), *puis celui des deux problèmes* (*a*), (*b*), *non résolus et aller en* 2.

Nous nommerons DAK le premier des deux algorithmes, et BBB le second. Tous deux, comme celui de Land et Doig d'ailleurs, s'appliquent au cas où le minimum de (1) est une fonction convexe et où la région définie par (2) est convexe. Ils peuvent également s'appliquer dans des cas plus étendus de la programmation non linéaire. Si cependant le problème PP peut avoir des cols ou des minimums locaux non absolus, les procédures exposées deviennent alors « heuristiques, c'est-à-dire, dans l'acception relativement récente de cette dernière locution, des procédures donnant une assez bonne chance de trouver une solution qui ne soit pas trop mauvaise.

Pour mémoriser un problème, en programmation non linéaire, il suffit de mémoriser la solution optimale x_{PP} du problème PP dont il descend, plus quelques informations occupant à peine deux ou trois mémoires. Si l'on mémorise x_{PP} en effet, passer de x_{PP} à la résolution de (*a*), (*b*) ou (*c*) ne requiert qu'un très petit nombre d'itérations (deux ou trois en moyenne). Si l'on n'utilise pas x_{PP} comme point de départ, ce nombre d'itérations est aisément multiplié par 10, voire parfois par 100.

Ces considérations expliquent que, en 4 ou 4', la création de *deux* problèmes n'exige en fait que la réservation d'un seul groupe de mémoire pour x_{PP}, à deux ou trois unités près.

EXPÉRIENCES NUMÉRIQUES

L'expérimentation numérique a porté sur sept programmes non linéaires entiers et quatre critères de choix au pas 4 ou 4' pour DAK et BBB respectivement. Le critère de choix porte sur la variable x_β et sur l'ordre des problèmes (*a*) ou (*b*) à ajouter à la liste. Comme le dernier entré sera résolu immédiatement après, on peut dire que l'on choisit en fait simultanément

une variable x_β, dite « arbitrée », et un des deux problèmes à résoudre immédiatement, l'autre étant mis en attente dans la liste.

Soit x_j^* $j \in E$, une valeur non entière, et soient x' et x'' respectivement les vecteurs obtenus à partir de x^* en y remplaçant respectivement x_j^* par $[x_j^*]$ et $\langle x_j^* \rangle$. Posons $A_j = \varphi(x'_j)$, $B_j = \varphi(x''_j)$; A_j, B_j sont appelés *pénalités*. Voici les quatre critères :

CRITÈRE 1 : *Arbitrage au plus proche d'un entier :* x_β^* est celui des x_j^*, $j \in E$, qui est le plus proche d'un entier (sans être un entier). Le problème que l'on choisit de traiter correspond à celui des deux nombres $[x_\beta^*]$, $\langle x_\beta^* \rangle$, qui est le plus proche de x_β^*.

CRITÈRE 2 : *Arbitrage au plus éloigné d'un entier* : x_β^* est celui des x_j^*, $j \in E$, qui est le plus éloigné d'un entier. Le problème que l'on choisit de traiter est le même que pour le critère 1.

CRITÈRE 3 : *Mise en attente du problème correspondant à la plus grande pénalité :* si A_β est le plus grand de tous les A_j, B_j, $j \in E$, cela détermine β; on choisit l'ordre (a), (b). Procéder de même si B_β est le plus grand de tous les A_j, B_j, $j \in E$, mais choisir alors l'ordre (b), (a).

CRITÈRE 4 : *Résolution du problème correspondant à la plus petite pénalité :* si A_β est le plus petit de tous les A_j, B_j, $j \in E$, cela détermine β; on choisit l'ordre (b), (a). Procéder de même si B_β est le plus petit de tous les A_j, B_j, $j \in E$, mais choisir alors l'ordre (a), (b).

Les sept problèmes-test que nous avons utilisés se trouvent dans les références suivantes, auxquelles le lecteur intéressé est renvoyé :

Problème 1 : Abadie [1].
Problème 2 : Bracken et McCormick [6].
Problème 3 : Rosen et Suzuki [17].
Problème 4 : Hammer et Rudeanu [11].
Problème 5 : Colville [7] (problème 1).
Problème 6 : Colville [7] (problème 3).
Problème 7 : Mao [16] (p. 295).

Tous ces problèmes, sauf le problème 4, sont totalement entiers. Aucun de ces problèmes n'est linéaire; deux ne sont pas convexes (5 et 6). On trouve, pour chaque problème, la même solution avec les dix essais, sauf en ce qui concerne le problème 6, pour lequel on trouve un minimum égal à 30 452,93 lorsqu'on applique BBB ($x = 78; 33; 31; 45; 35$) et 30 512,45 avec DAK ($x = 81; 33; 30; 45; 36$). L'un de ces deux minimums au moins est donc local, et BBB semble donner une meilleure chance d'obtenir une meilleure solution entière.

La méthode d'optimisation non linéaire sous-jacente (résolution de PP) est la méthode GRG (Abadie et Carpentier [3]), classée première dans le classement de Colville (Colville, [8]). On pourra en trouver un exposé, par exemple, dans le livre de Himmelblau [12].

Dans le tableau qui suit, on désigne par DAK i ou BBB i respectivement l'algorithme DAK ou BBB avec le critère $i = 1, 2, 3, 4$. Le nombre S_k (le « score ») est la valeur de

$$S_k = \frac{1}{7} \sum_{j=1}^{7} \frac{t_{kj}}{\min_l (t_{lj})},$$

où t_{kj} est le temps nécessaire pour résoudre le problème j avec la méthode k. La méthode est d'autant meilleure que S_k est plus petit, l'idéal étant 1. En gros, le rapport des temps de calcul pour deux méthodes k et k' ne devrait pas être trop éloigné de S_k/S_k' (Abadie et Guigou [4]) :

Méthode	S	Méthode	S
BBB 1	1.16	DAK 3	1.85
BBB 3	1.18	DAK 1	3.17
BBB 4	1.36	DAK 2	3.22
BBB 2	1.37	DAK 4	3.67

CONCLUSION

Il semble résulter du tableau précédent que BBB soit supérieur à DAK en ce qui concerne la vitesse d'exécution, ce qui n'était nullement évident *a priori*. BBB pourrait donner aussi, comme on l'a vu, une meilleure chance de réussite dans le cas d'un problème non convexe. Enfin, elle possède la qualité que le nombre de mémoires nécessaires à la mémorisation de l'arborescence peut être borné *a priori*.

Si l'on avait à choisir entre les critères, il faudrait peut-être préférer 1 et 3 pour BBB, et 3 en ce qui concerne DAK.

L'expérience faite est encourageante, en ce sens que les problèmes se résolvent assez vite (86 secondes pour les 56 essais sur une machine CDC 6600 sous SCOPE 3 *b*), et que certaines différences paraissent assez significatives. Ce travail devrait être poursuivi, pour atteindre des conclusions plus solides, avec un plus grand nombre de problèmes-test, d'autres critères, plus fins, et peut-être la seconde des deux méthodes BBB mentionnées dans la référence [1].

BIBLIOGRAPHIE

1. J. ABADIE, *Une méthode arborescente pour les programmes partiellement discrets*, R.I.R.O., 3e année, V 3, 1969, p. 24-50.
2. J. ABADIE, *Une méthode de résolution des programmes non linéaires partiellement discrets sans hypothèse de convexité*, R.I.R.O., 5e année, V 1, 1971, p. 23-38.

octobre 1976.

3. J. ABADIE et J. CARPENTIER, *Generalization of the Wolfe Reduced Gradient Method to the Case of Nonlinear Constraints*, in Optimization, R. Fletcher, ed., Academic Press, New York, 1969.
4. J. ABADIE et J. GUIGOU, *Numerical Experiments with the GRG Method*, in Integer and Nonlinear Programming, J. Abadie, ed., North-Holland Publishing Company, Amsterdam, 1970.
5. E. M. L. BEALE et R. E. SMALL, *Mixed Integer Programming by a Branch and Bound Technique*, in Proceedings of the IFIP Congress 1965, p. 450-451, W. A. Kalenich, ed., Spartan Press, Washington D. C., 1965.
6. J. BRACKEN et G. P. MCCORMICK, *Selected Applications of Nonlinear Programming*, Wiley, New York, 1968.
7. A. R. COLVILLE, *A Comparative Study of Nonlinear Programming Codes*, IBM NYSC Report 320-2949, 1968.
8. A. R. COLVILLE, *A Comparative Study of Nonlinear Programming Codes*, p. 487-502, in Proceedings of the Princeton Symposium on Mathematical Programming H. W. Kuhn, ed., Princeton University Press, 1970.
9. R. J. DAKIN, *A Tree-Search Algorithm for Mixed Integer Problems*, The Computer Journal, vol. 8, 1965, p. 250-255.
10. N. J. DRIEBECK, *An Algorithm for the Solution of Mixed Integer Programming Problems*, Management Science, vol. 12, 1966, p. 576-587.
11. P. L. HAMMER et S. RUDEANU, *Méthodes booléennes en recherche opérationnelle*, Dunod, Paris, 1970.
12. D. M. HIMMELBLAU, *Applied Nonlinear Programming*, McGraw-Hill, New York, 1972.
13. A. H. LAND et A. G. DOIG, *An Automatic Method of Solving Discrete Programming Problems*, Econometrica, vol. 28, 1960, p. 497-520.
14. P. LE FAOU, *Une méthode arborescente pour la résolution des programmes linéaires partiellement en nombres entiers*, Thèse de 3e cycle, Université Paris VI, Paris, 1973.
15. J. D. C. LITTLE, K. C. MURTY, D. W. SWEENEY et C. KAREL, *An Algorithm for the Traveling Salesman Problem*, Operations Research, vol. 11, 1963, p. 972-989.
16. J. C. T. MAO, *Quantitative Analysis of Financial Decisions*, The MacMillan Company, Collier-MacMillan Limited, Londres, 1969.
17. J. B. ROSEN et S. SUZUKI, *Construction of Nonlinear Programming Test Problems*, Commun. A.C.M., vol. 8, 1965, p. 113.
18. B. ROY, R. BENAYOUN et J. TERGNY, *From S.E.P. Procedure to the Mixed OPHELIE Program*, in Integer and Nonlinear Programming, J. Abadie, ed., North-Holland Publishing Company, Amsterdam, 1970.
19. J. A. TOMLIN, *Branch and Bound Methods for Integer and Non-Convex Programming*, in Integer and Nonlinear Programming, J. Abadie, ed., North-Holland Publishing Company, Amsterdam, 1970.