REVUE FRANÇAISE D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE. SÉRIE VERTE

J. ABADIE

Une méthode arborescente pour les programmes non-linéaires partiellement discrets

Revue française d'informatique et de recherche opérationnelle. Série verte, tome 3, n° V3 (1969), p. 25-49

<http://www.numdam.org/item?id=RO_1969__3_3_25_0>

R.I.R.O.
(3e année, V-3, 1969, p. 24-50)

UNE METHODE ARBORESCENTE POUR LES PROGRAMMES NON-LINEAIRES PARTIELLEMENT DISCRETS

J. ABADIE
Electricité de France

Résumé. — *La « méthode BBB » (« Bounded Branch and Bound ») est exposée et illustrée d'un exemple numérique. Elle possède sur d'autres méthodes arborescentes les avantages suivants :* a) *le nombre des sommets de l'arborescence à conserver en mémoire est borné* a priori ; b) *cette borne est raisonnablement petite (de 1 à 2N — 2, ou N est le nombre des variables entières ;* c) *des solutions entières « acceptables » sont obtenues plus rapidement (avantage qui joue lorsque l'on est amené à arrêter les calculs avant achèvement). Les démonstrations sont données dans l'hypothèse de quasi-convexité.*

INTRODUCTION

Malgré la faveur dont jouissent, à juste titre, les méthodes arborescentes pour la résolution des problèmes d'optimisation discrète, la difficulté causée par la croissance incontrôlable de l'arborescence qu'elles engendrent ne semble pas avoir reçu de solution, non plus qu'une attention suffisante, même dans les publications récentes (Hervé, 1968 ; Roy, Benayoun et Tergny, 1970). Bien entendu, l'utilisation des bandes permet aux ordinateurs de surmonter cette difficulté. Bien des problèmes cependant, comportant quelques centaines de variables (entières ou non) et un nombre de contraintes du même ordre de grandeur (auxquelles s'ajoutent les conditions de positivité, ou plus généralement de limitations des variables), pourraient être résolus *entièrement en mémoire centrale* si la partie utile de l'arborescence pouvait être conservée dans les limites de, disons, 1 à $2N$ sommets, où N est le nombre des variables x_j qui doivent prendre des valeurs discrètes (le nombre de ces valeurs n'étant d'ailleurs pas limité).

Cette difficulté a d'ailleurs favorisé l'éclosion d'une foule de méthodes dites « heuristiques ». Dans le contexte présent, ce dernier mot qualifie

Nous tenons à remercier MM. F. Ceschino et J. Mariotti : les discussions que nous avons eues ensemble ont été des plus utiles à la mise au point de cet exposé.

les méthodes où aucune certitude d'atteindre la solution optimale n'existe, mais qui construisent cependant des points réalisables ayant « intuitivement » parlant, une « bonne probabilité » d'être « proche » de la solution optimale. La plupart du temps, les mots entre guillemets ne sont d'ailleurs pas définis. Ces méthodes heuristiques ont naturellement une certaine valeur, puisqu'il faut bien se contenter de « presque résoudre » un problème lorsque sa résolution est soit impossible soit trop coûteuse. Citons trois de ces procédures heuristiques :

1. Résoudre le problème d'optimisation « continu » correspondant au problème posé, puis arrondir « au mieux », dans la solution obtenue, les variables qui auraient dû être entières, (*continu* signifie souvent dans cet exposé, *non-discret*) ; optimiser ensuite, par rapport aux variables non entières, le problème obtenu lorsqu'on fixe, dans le problème original, les variables entières aux valeurs que l'on vient d'obtenir (*problème réduit* correspondant à ces valeurs entières).

2. Commencer par la procédure heuristique ci-dessus, puis modifier d'une unité, en plus ou en moins, la valeur d'une variable entière, et résoudre le problème réduit correspondant ; répéter cette procédure avec la même variable entière, tant que $\varphi(x)$, la fonction à minimiser, diminue, puis passer à la modification d'une seconde variable entière ; arrêter la procédure en un point tel que les modifications de $\pm$ 1 de chaque variable entière, *prise séparément*, ne permettent plus d'abaisser $\varphi(x)$; on court le risque de négliger des modifications simultanées qui pourraient diminuer $\varphi(x)$, mais, s'il y a N variables entières, le nombre de ces modifications, qui est 3^N, peut être trop grand. En outre, la considération de toutes ces modifications ne suffirait pas à rendre la méthode rigoureuse.

3. Utiliser une procédure arborescente, en limitant à N' le nombre des sommets de l'arborescence à mémoriser ; lorsque le nombre des sommets devient supérieur à N', on supprime le sommet « le plus mauvais » (en acceptant de courir le risque que ce soit dans la descendance d'un tel sommet que se trouve la solution optimale, et de ne point la trouver).

D'autres procédures heuristiques, beaucoup plus élaborées, sont aussi en usage. Elles seraient trop longues à exposer, et nous feraient sortir du cadre de cet article. Lorsqu'elles sont attirantes, elles ont en commun de donner assez rapidement une solution discrète qui, même si elle n'est pas optimale, devrait être « acceptable dans la pratique ».

L'expérience acquise à ce jour dans les méthodes exactes (par opposition aux méthodes heuristiques) concerne surtout les problèmes discrets *linéaires*. Il est indispensable de quitter, même pour un code général expérimental, ce domaine linéaire. Ceci est parfaitement réalisable : on pourra trouver dans un rapport de Colville (1968) des tests de comparaison sur machine concernant une trentaine de méthodes d'optimisation non-linéaire *continue* (pour une autre comparaison, voir Abadie et Guigou, 1970 ; pour la méthode GRG, classée en tête par Colville, voir Abadie, et Carpentier 1966 et 1969 ; Abadie et Guigou, 1969 ; Abadie, 1969 ; pour une description de diverses méthodes, outre Colville, 1968, voir Abadie, 1967, Fletcher, 1969 ; Abadie, 1970). Il existe d'ailleurs des

méthodes arborescentes différentes de celle qui fait l'objet de cet exposé couplées avec une méthode générale de résolution des programmes non-linéaires (Carpentier, 1969), et même des codes réalisés pour des problèmes particuliers (Carpentier, Cassapoglou et al. 1968).

La méthode arborescente que nous allons maintenant exposer (et que nous avons baptisée BBB, pour « Bounded Branch and Bound ») est d'une variété parente de celle que Land et Doig (1960) ont proposée pour la programmation linéaire partiellement discrète. Elle entre dans l'axiomatique de Hervé (1968) et celle de Roy (1969) qui semblent bien recouvrir toutes les procédures arborescentes connues. Elle possède les propriétés fondamentales dont nous croyons avoir fait sentir la nécessité et la réalisabilité :

1. le nombre des sommets de l'arborescence qu'il est nécessaire de conserver en mémoire est faible ($2N - 2$), ou même 1 ;

2. elle s'applique au cas où la fonction économique et les contraintes ne sont pas linéaires (en utilisant, comme sous-programme, un quelconque des codes de programmation non-linéaire existant) ;

3. elle possède la propriété fondamentale, bien que « mal définie », des procédures heuristiques, à savoir de fournir assez rapidement une première solution discrète « acceptable ».

Nous utiliserons dans la suite un langage imagé (et même des images). L'exposé est destiné à être compris par des utilisateurs non familiarisés avec des procédures arborescentes. Un exemple numérique entièrement traité illustre la méthode : cet exemple n'est pas choisi pour faire briller la méthode en minimisant le nombre des itérations ; il s'agit d'un exemple difficile, malgré les apparences. La plupart des méthodes heuristiques échouent à donner une solution raisonnablement bonne, et le nombre des itérations pour notre algorithme y est même anormalement élevé. Tel quel, cependant, cet exemple illustre la plupart des phénomènes que l'on peut rencontrer.

1. LE PROBLEME P ET LE GRAPHE G SOUS-JACENT

Considérons le programme partiellement discret suivant :

minimiser $\varphi(x)$

sous les contraintes :

$$x \in K$$
$$x_j \in F_j, \forall j \in E$$
$$a_j \leqslant x_j \leqslant b_j, \forall j \in E',$$

où l'on a adopté les notations suivantes :

x est le vecteur $(x_j)_{j \in J}$, $J = \{1, ..., n\}$;
K est une partie de l'espace R^n ;
E est une partie de J ; $E' = J - E$;
F_j est un ensemble fini de nombres réels.

La méthode que nous allons exposer résoud le problème précédent, sous certaines hypothèses sur φ et K. L'exposé est plus simple pour le cas particulier suivant, sans qu'il y ait, pour la méthode, perte de généralité. Nous supposerons donc que le problème posé est le problème partiellement entier :

P : minimiser $\varphi(x)$

sous les contraintes :

$$x \in K \tag{1}$$

$$a_j \leqslant x_j \leqslant b_j, \forall j \in J \tag{2}$$

$$x_j \text{ entier}, \quad \forall j \in E ; \tag{3}$$

on supposera a_j, b_j entiers, $\forall j \in E$, ce qui ne diminue pas la généralité.

Pour la commodité des démonstrations, le parallèlotope Π défini par (2) est supposé borné (ce qui n'est jamais une gêne dans la pratique), et la fonction φ est supposée strictement quasi-convexe sur Π. Rappelons pour mémoire que φ est dite strictement quasi-convexe sur Π si le maximum de φ sur un segment quelconque contenu dans Π ne peut pas être atteint en un point intérieur au segment. Pour citer l'exemple le plus simple, toute fonction convexe n'atteignant pas son minimum en plusieurs points est strictement quasi-convexe (il en est ainsi, en particulier, des fonctions linéaires non identiques à une constante). L'ensemble K est supposé convexe.

Soit A une partie quelconque $A \subset E$, et $x_A = (x_j)_{j \in A}$ le sous-vecteur de x dont les composantes ont pour indices j les éléments de A. Dans la suite, $\bar{x}_A$, $\tilde{x}_A$, $\check{x}_A$, désignent des vecteurs x_A dont les composantes sont entières, et *fixées* à des valeurs vérifiant (2).

A la donnée d'une paire $S = (A, \bar{x}_A)$ on fait correspondre le problème $P(S)$ défini comme suit :

$P(S)$: minimiser $\varphi(x)$

sous les contraintes :

$$x \in K \tag{1}$$

$$a \leqslant x \leqslant b \tag{2}$$

$$x_j = \bar{x}_j, \forall j \in A. \tag{4}$$

Soit $\hat{x}(S)$ la solution de $P(S)$ (supposée unique pour simplifier), et $\hat{\varphi}_S$ la valeur minimale de φ dans $P(S)$, avec la convention usuelle : $\hat{\varphi}_S = +\infty$ si (1), (2), (4) sont incompatibles. Il résulte de ce qui précède que $\bar{x}_j = \hat{x}_j(S)$, $\forall j \in A$. Soit $E(S)$ l'ensemble des indices $j \in E$ tels que $\hat{x}_j(S)$ soit entier.

On considère le graphe orienté G défini de la façon suivante :

1. Les sommets de G sont les diverses paires $S = (A, \bar{x}_A)$ définies ci-dessus.

2. A chaque sommet S est attaché sa *valeur* $\hat{\varphi}_S$.

3. Lorsque A est vide, le sommet correspondant est appelé la *racine* S_0 de G.

4. Lorsque $E(S) = E$ on dit que le sommet $S = (A, \bar{x}_A)$ est *terminal*.

5. A tout sommet $S = (A, \bar{x}_A)$ on fait correspondre son *étage* ts, qui est le nombre des éléments de A (l'étage correspond au *rang* dans la terminologie de Roy, 1969).

6. Soit $S = (A, \bar{x}_A)$ un sommet non-terminal, et soit $\beta \in E - A$. On définit un successeur d'étage supérieur de S relatif à β par

$$T = (B, \tilde{x}_B),$$

où :

$$B = A \cup \{ \beta \}$$
$$\tilde{x}_j = \bar{x}_j, \forall j \in A$$
$$\tilde{x}_\beta = [\hat{x}_\beta(A)] \quad \text{ou} \quad [\hat{x}_\beta(A)] + 1$$

(la notation $[\lambda]$ désigne, suivant l'usage, la partie entière du nombre réel λ). Il existe deux successeurs de S relativement à β.

7. Plus généralement, étant donné le sommet non-terminal $S = (A, \bar{x}_A)$, considérons les sommets d'étage $ts + 1$ définis par :

$$\beta \in E - A$$
$$B = A \cup \{ \beta \}$$
$$\tilde{x}_j = \bar{x}_j, \forall j \in A$$
$$\tilde{x}_\beta = [\tilde{x}_\beta(A)] + \gamma$$
$$T = (B, \tilde{x}_\beta),$$

où γ est un entier nul ou de signe quelconque, pourvu que (2) soit vérifié.

On joint par des arcs successifs les sommets correspondant à $\gamma = 0, -1, -2, \ldots$, qui constituent *l'aile gauche* dont *l'origine* est le sommet correspondant à $\gamma = 0$. On joint également par des arcs successifs les sommets correspondant à $\gamma = 1, 2, \ldots$, qui constitue *l'aile droite* dont l'origine est le sommet correspondant à $\gamma = 1$. Les deux ailes sont dites *opposées* et *issues* de S.

L'éloignement d'un sommet dans son aile est $|\gamma|$ pour une aile gauche, $\gamma - 1$ pour une aile droite.

Il est bon de remarquer que, lorsque $\beta \in E(S) - A$ et $\gamma = 0$, on a, *sans aucun calcul :* $\hat{x}(T) = \hat{x}(S)$, $\hat{\varphi}_S = \hat{\varphi}_T$.

EXEMPLE : Soit le problème :

minimiser

$$2(x_1^2 + x_2^2 + x_3^2) - (x_1 + x_2 + 9x_3)$$

sous les contraintes :

$$-2 \leqslant x_j \leqslant 2, \forall j \in J = \{1, 2, 3\}$$
$$x_j \text{ entier}, \quad \forall j \in E = J.$$

Posons $A = \{1\}, \bar{x}_1 = 2$ pour définir le sommet $S = (A, \bar{x}_A)$. La solution de $P(S)$ est : $\hat{x}(S) = \left(2, \frac{1}{4}, 2\right)$.

Le sous-graphe partiel de G composé de $S = (A, \bar{x}_A)$, de ses successeurs d'étage 2 relatifs à $\beta = 2$, et des deux ailes opposées correspondantes est donnée par la figure 1 *a*, qui sera toujours dessinée sous la forme de la figure 1 *b*, où les flèches indiquant le sens des arcs sont sous-entendues.

On a posé :

$$T_i = (B_i, \tilde{x}^i_{B_i}),$$

avec :

$$B_i = \{1, 2\}, i = 1, ..., 5$$
$$(\tilde{x}^i_1, \tilde{x}^i_2) = \begin{cases} (2, 1 - i), i = 1, 2, 3 \\ (2, i - 3), i = 4, 5. \end{cases}$$

T_1 et T_4 sont origines de deux ailes opposées. L'éloignement de T_i est :

$$i - 1, i = 1, 2, 3$$
$$i - 4, i = 4, 5.$$

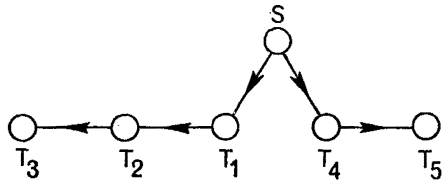


Figure 1 *a*

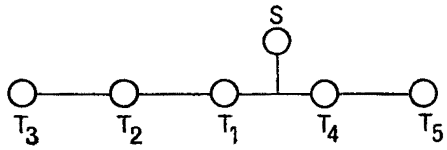


Figure 1 *b*

Dans le graphe G, outre la notion de « successeur » (T est un successeur de S lorsque ST est un arc ; successeur a le même sens que « suivant »

dans Roy, 1969), il existe la notion de *descendant* (Roy, 1969) ; T est un descendant de S s'il existe un chemin joignant S à T (un successeur est un descendant tel qu'il existe un chemin formé d'un seul arc).

Théorème 1. Si T est un descendant de S dans G, alors $\hat{\varphi}_T \geqslant \hat{\varphi}_S$.

Démonstration. Reprenons les notations de la section 6 de la définition de G, et supposons que T appartienne à une aile droite directement issue de S. Le segment $(\hat{x}_S, \hat{x}_T)$ coupe les hyperplans $x_\beta = [\hat{x}_\beta(A)] + \gamma$ en des points x^γ tel que $\varphi(x^\gamma)$ croisse avec γ. Il en résulte tout d'abord que $\hat{\varphi}_T \geqslant \hat{\varphi}_S$, ce qui achève la démonstration si $\gamma = 1$; on voit, lorsque $\gamma > 1$, en considérant T' correspondant à la valeur $\gamma - 1$, que $\hat{\varphi}_{T'} \leqslant \hat{\varphi}_T$: c'est dire que $\hat{\varphi}_T$ croît lorsque, dans l'aile droite, l'éloignement croît. Le même raisonnement s'applique à l'aile gauche.

Théorème 2. Le graphe G n'a pas de circuit.

Démonstration. Le rang d'un sommet ne peut que croître ou rester constant. Il ne reste constant que dans les ailes, qui n'ont pas de circuit.

2. EXPOSE DE LA METHODE

La méthode de Land et Doig consiste, comme le lecteur pourra le constater (Land et Doig, 1960 ; Balinski, 1967 ; Hervé, 1968) à construire, d'une certaine façon, un chemin joignant, dans le graphe G précédent, la racine S_0 au sommet terminal de valeur minimum. La construction de Land et Doig possède un défaut important dans les applications : il est impossible de donner *a priori*, une borne raisonnable du nombre des sommets de l'arborescence qu'il est nécessaire de conserver dans les mémoires de la machine aux étapes intermédiaires. Or mémoriser un sommet S revient, à quelques unités près, à conserver $\hat{x}(S)$, ce qui exige n mémoires.

Bien entendu, on pourrait se contenter d'explorer tous les sommets terminaux de G, et d'en retenir celui qui a la plus petite valeur (dans le cas où $E = J$, cela reviendrait à calculer la valeur de $\varphi(x)$ pour $p = p_1 \times p_2 \times \ldots \times p_n$ points, où $p_j = [b_j - a_j]$, par exemple $p = 10^n$ si $p_j = 10$, $\forall j$). Il existe d'ailleurs des cas où c'est là la meilleure façon d'opérer (exemple : $E = \{1\}$, $a_1 = 0$, $b_1 = 1$: la procédure de Land et Doig ou celle qui suit conduisent à trois optimisations, l'exploration exhaustive des sommets terminaux à deux seulement parmi ces trois).

Dans ce qui suit est indiquée une méthode itérative qui construit dans G une arborescence (Berge, 1958 ; Hervé, 1968 ; Roy, 1969) ; en ce qui nous concerne, une *arborescence* sera ici un sous-graphe partiel de G contenant la racine S_0, et tel que tout sommet autre que S_0 soit l'extrémité d'un arc et d'un seul. A chaque sommet du graphe est attaché un certain nombre de *marques* qui seront précisées (voir exemple illustratif).

A chaque début d'itération, l'arborescence est dans un certain état, et on possède une estimation par excès, héritée des itérations précédentes, de la valeur minimale de φ dans P. Cette estimation est appelée $\hat{\varphi}$ ($\hat{\varphi} = +\infty$ au début de la première itération, sauf si l'on connaissait une solution des contraintes (1), (2), (3) : on prendrait alors pour $\hat{\varphi}$ la valeur de φ pour cette solution).

Examiner un sommet $S = (A, \bar{x}_A)$, c'est résoudre $P(S)$, c'est-à-dire :

soit trouver que les contraintes (1), (2), (4) de $P(S)$ sont incompatibles, et donner à S la valeur $\hat{\varphi}_S = +\infty$;

soit trouver la solution $\hat{x}(S)$ de $P(S)$, et donner à S la valeur $\hat{\varphi}_S = \varphi(\hat{x}(S))$.

Un sommet S est *refusé* si $\hat{\varphi}_S \geqslant \hat{\varphi}$, ou si tous les descendants de S ont une valeur supérieure ou égale à $\hat{\varphi}_S$. D'après le théorème 1, tout descendant d'un sommet refusé est refusé. Un sommet refusé n'est jamais conservé en mémoire (voir aussi la règle R 6 plus loin).

Après son examen, tout sommet est, soit accepté, soit refusé. Il est *accepté* chaque fois qu'il n'est pas refusé. Cette acceptation est provisoire, car $\hat{\varphi}$ diminue d'itération en itération, ce qui fait qu'un sommet accepté à une certaine itération peut être refusé à une itération ultérieure (le refus, lui, est définitif, car aucun sommet terminal qui serait descendant d'un sommet refusé ne saurait être optimal pour le problème original P). Un sommet accepté est toujours conservé en mémoire, jusqu'à son refus éventuel.

On emploiera le langage graphique suivant :

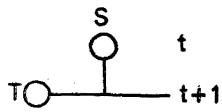
sommet accepté (provisoirement), d'étage t ;

sommet refusé (définitivement, ainsi que tous ses descendants), d'étage t ;

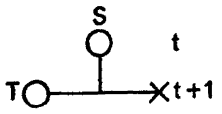
sommet accepté (provisoirement) à l'étage t dont les descendants d'étage $> t$ sont refusés ;

sommet accepté (provisoirement) d'étage t dont les descendants d'éloignement plus grand de la même aile (gauche ici) sont refusés ;

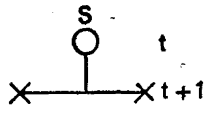
même signification (aile droite au lieu d'aile gauche) ;



le sommet S, d'étage t est accepté (provisoirement) ainsi que le sommet T, d'étage $t+1$; le sommet T est un descendant de S dans G (il n'est pas nécessairement un successeur de S dans G, car son éloignement peut être grand) appartenant à une aile gauche issue de S ; aucun sommet de l'aile opposée n'a été examiné ;



même interprétation que ci-dessus, mais l'aile opposée est refusée (ainsi que les descendants de tous les sommets figurant dans cette aile) ;



le sommet S est accepté (provisoirement) à l'étage t; deux ailes opposées sont refusées à l'étage $t + 1$.

A chaque état de l'arborescence correspond un certain schéma graphique. Exemples avec quatre étages :

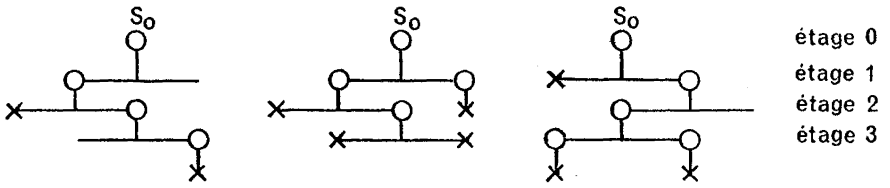


Figure 2

Le numéro de l'étage le plus élevé, désigné par t_m, est toujours inférieur ou égal au nombre des éléments de E. Le schéma est toujours accompagné, pour chaque sommet accepté $S = (A, \bar{x}_A)$, de sa valeur $\hat{\varphi}_S$ et de la solution $\hat{x}(S)$ de $P(S)$; une série de mémoires est en outre réservée à $\hat{\varphi}$, et à la meilleure solution $\hat{x}$ de (1), (2), (3) obtenue au cours des itérations précédentes dès que l'on en possède une.

L'algorithme proposé comporte un certain nombre de règles, que nous allons maintenant indiquer. Ces règles (R 0 à R 6) sont à appliquer en succession, et sont schématiquement illustrées plus loin sous les noms d'états 0, 1, ..., 9, relatifs à l'étage le plus élevé t_m. Il ne peut exister, comme on le verra, dans chaque étage plus de deux sommets acceptés. S'il y en a deux, ils appartiennent à deux ailes opposées.

Règles de l'algorithme

R 0. Considérer l'étage le plus élevé t_m (pour la première itération, $t_m = 0$, examiner l'origine S_0).

R 1. Si les descendants d'étage supérieur de S_0 sont tous refusés, FIN du problème : les mémoires réservées à la meilleure solution de (1), (2), (3) contiennent la solution de P si une solution de (1), (2), (3) y a été placée à un moment quelconque ; sinon, les contraintes (1), (2), (3) sont incompatibles.

R 2. S'il existe dans l'étage t_m un sommet accepté T dont les descendants d'étage supérieur ne sont pas tous refusés, examiner un successeur. U de T dans l'étage $t_m + 1$, et accepter ou refuser U. L'étage t_m se trouve augmenté d'une unité (états 1, 2, 3, 4).

R 3. Sinon, lorsque cela est possible, examiner, dans l'ensemble des deux ailes opposées qui constituent t_m, le sommet dont l'éloignement est le plus petit, et l'accepter ou le refuser (états 5, 6, 7, 8). Si ce dernier sommet est le successeur d'un autre dans la même aile, ce dernier est effacé (états 7, 8).

R 4. Si deux ailes opposées de l'étage $t_m > 0$ sont refusées, refuser également tous les successeurs d'étage supérieur du sommet S dont sont issues ces deux ailes opposées. L'étage t_m se trouve diminué d'une unité (état 9).

R 5. A la fin de l'itération, on a :

$$\hat{\varphi} \text{ nouveau} \leqslant \hat{\varphi} \text{ ancien ;}$$

lorsque

$$\hat{\varphi} \text{ nouveau} < \hat{\varphi} \text{ ancien,}$$

passer en revue tous les $\hat{\varphi}_S$ conservés en mémoire, et refuser (avec effacement des mémoires) tous les sommets S pour lesquels :

$$\hat{\varphi}_S \geqslant \hat{\varphi} \text{ nouveau.}$$

R 6. Si $\hat{x}_j(S)$ est entier pour tout $j \in E$, alors S est refusé. Si en outre $\hat{\varphi}_S < \hat{\varphi}$, alors $\hat{x}(S)$ devient la meilleure solution courante des contraintes de P, et remplace la précédente dans les mémoires réservées à cet effet, tandis qu'on fait $\hat{\varphi} := \hat{\varphi}_S$. Si au contraire $\hat{\varphi}_S \geqslant \hat{\varphi}$, alors l'ancienne meilleure solution courante reste en place.

Les règles R 1 à R 4 se résument à une transformation du dernier étage, schématisée ci-après. Les règles R 5 et R 6 permettent de diminuer le nombre des sommets conservés en mémoire, elles ne sont pas schématisées ci-après. Ri Acc. (Ri Ref.) signifie que le sommet examiné en application de la règle Ri est accepté (refusé).

Pratique d'un passage d'un état à un autre

1. Lorsque l'étage t_m augmente d'une unité (états n^{os} 1, 2, 3, 4), c'est qu'une variable x_β change de valeur. On part de $\hat{x}(T)$, et l'on maximise (ou minimise) x_β, jusqu'à ce que x_β atteigne la valeur désirée (attribuer dans (2), pour ce faire, à x_β la borne supérieure ou inférieure égale à la valeur désirée). Le résultat de cette maximisation de x_β est un point vérifiant les contraintes de $P(T)$ (s'il en existe ; sinon, on conclura que $\hat{\varphi}_T = +\infty$).

2. Lorsque l'étage t_m reste le même, on procède de façon analogue, soit à partir de S (état 5), soit à partir de T' (états 6, 7, 8).

3. Lorsque l'étage t_m diminue (état 9 ou application de la règle R 6), la question d'obtenir un premier point vérifiant certaines contraintes ne se présente pas.

3. RESULTATS THEORIQUES

Nous allons maintenant montrer que l'algorithme possède bien les propriétés indiquées sur le nombre des mémoires à réserver (théorème 4) et sur la convergence (théorème 5). On démontrera auparavant le théorème 3 qui est utile tant pour la démonstration des deux autres que pour faciliter la programmation de la méthode sur machine.

Théorème 3. Dans l'application de l'algorithme, l'étage le plus élevé est, au début de toute itération, dans un des états 0 à 9 ci-dessus, les étages moins élevés dans un des états 0, 1, 2, 3, 4, 6, 7, 8.

Démonstration. Chacun des états 0 à 9 ne peut engendrer, par les règles R 1 à R 4, qu'un état 0 à 9. Les cas 5, 9 sont exclus pour les étages inférieurs qui doivent posséder un point accepté. L'application de R 5, R 6 ne change pas la conclusion, d'après le théorème 1.

Théorème 4. A toute itération, l'état de l'arborescence ne compte pas plus de $2N - 1$ sommets acceptés, où N est le nombre des variables entières (c'est-à-dire le nombre des éléments de E).

D'après le théorème 3, ce nombre est au plus $2N + 1$; il se réduit à $2N - 1$ par l'application de la règle R 6.

REMARQUES

1. Le nombre maximum de sommets figurant dans l'arborescence peut même être considéré comme étant $2N - 2$, si l'on remarque que le sommet racine S_0 devient inutile dès que l'examen de deux ailes opposées issues de S_0 a été débuté.

2. Ce nombre maximum est atteint lorsque $t_m = N - 1$ et que les étages de 1 à $N - 2$ sont dans l'état 4, l'étage $N - 1$ étant dans l'état 4 ou 8.

Théorème 5. En un nombre fini d'itérations, l'algorithme trouve une solution de P, ou bien conclut que les contraintes de P sont incompatibles.

Démonstration. Le nombre des itérations est fini parce que le nombre des sommets de G est fini, et que l'on ne rencontre jamais deux fois le même.

Les deux fins possibles sont :

FIN 1 : État 0 *et* $\hat{\varphi} < +\infty$ (problème résolu) ;

FIN 2 : État 0 *et* $\hat{\varphi} = +\infty$ (contraintes incompatibles).

Dans tous les cas, les sommets $S = (A, \bar{x}_A)$ qui ont été refusés l'ont été, soit parce qu'ils correspondaient à des problèmes $P(S)$ dont les contraintes étaient incompatibles (ce qui montre que $\bar{x}_A$ ne peut faire partie d'une solution des contraintes de P), soit parce que $\hat{\varphi}_S \geqslant \hat{\varphi} \geqslant \min \{ \varphi, \text{problème } P \}$. D'après le théorème 1, il en est de même de tout sommet terminal descendant de S. Il en résulte que les FINS 1 et 2 ont bien les significations indiquées.

Cas où le problème P est convexe et différentiable

Supposons que la fonction φ soit convexe et différentiable dans le parallélotope Π, et que l'ensemble convexe K soit défini par les inégalités :

$$f_i(x) \leqslant 0, \; i \in \{ 1, \ldots, m \},$$

où les fonctions f_i sont elles-même convexes et différentiables dans Π. Soit $U = (B, \breve{x}_B)$ le successeur de $T = (B, \tilde{x}_B)$ dans une même aile issue de $S = (A, \bar{x}_A)$. Rappelons que :

$$B = A \cup \{ \beta \} \qquad \text{où} \qquad \beta \in E - A$$

$$\tilde{x}_j = \breve{x}_j = \bar{x}_j, \; \forall \, j \in A$$

$$\tilde{x}_\beta = [\hat{x}_\beta(S)] + \gamma$$

$$\breve{x}_\beta = \tilde{x}_\beta + \varepsilon$$

$$\varepsilon = \begin{cases} + 1 \text{ pour une aile droite} \\ - 1 \text{ pour une aile gauche.} \end{cases}$$

La résolution de $P(T)$ fournit des multiplicateurs $\lambda_i(T)$, $\mu_k(T)$ tels que :

$$\left(\frac{\partial \varphi}{\partial x_k} + \sum_{i=1}^{m} \lambda_i(T) \, \frac{\partial f_i}{\partial x_k} \right)_T = \mu_k(T), \; k \in J - B$$

$$\lambda_i(T) \geqslant 0, \; \lambda_i(T) f_i(\hat{x}(T)) = 0, \; i = 1, \ldots, m$$

$$\mu_k(T) \geqslant 0 \qquad \text{si} \qquad \hat{x}_k(T) = a_k$$

$$\mu_k(T) \leqslant 0 \qquad \text{si} \qquad \hat{x}_k(T) = b_k$$

$$\mu_k(T) = 0 \qquad \text{si} \qquad a_k < \hat{x}_k(T) < b_k,$$

où la notation $(\;)_T$ sert à indiquer que l'expression entre parenthèses est à calculer pour $x = \hat{x}(T)$.

L'inégalité de convexité :

$$\varphi(\hat{x}(U)) + \sum_{i=1}^{m} \lambda_i(T) f_i(\hat{x}(U)) \geqslant \varphi(\hat{x}(T)) + \sum_{i=1}^{m} \lambda_i(T) f_i(\hat{x}(T))$$
$$+ \sum_{k \in J-B} \left(\frac{\partial \varphi}{\partial x_k} + \sum_{i=1}^{m} \lambda_i(T) \frac{\partial f_i}{\partial x_k} \right)_T (\hat{x}(U) - \hat{x}(T))$$
$$+ \varepsilon \left(\frac{\partial \varphi}{\partial x_\beta} + \sum_{i=1}^{m} \lambda_i(T) \frac{\partial f_i}{\partial x_\beta} \right)_T$$

donne :

$$\hat{\varphi}_U \geqslant \hat{\varphi}_T + \varepsilon \left(\frac{\partial \varphi}{\partial x_\beta} + \sum_{i=1}^{m} \lambda_i(T) \frac{\partial f_i}{\partial x_\beta} \right)_T = \tilde{\varphi}_T.$$

Cette relation permet de refuser U, sans avoir à l'examiner plus avant, lorsque $\tilde{\varphi}_T \geqslant \hat{\varphi}$. Si cette relation n'est pas vérifiée, on conservera cependant $\tilde{\varphi}_T$ en mémoire en même temps que T : comme $\hat{\varphi}$ ne peut que diminuer, il se peut ainsi que l'on soit à même de refuser U à une itération ultérieure.

D'autre part, dans la résolution de $P(S)$, les inégalités de convexité donnent une *borne inférieure* de $\hat{\varphi}_S$ à chaque itération utilisée pour résoudre $P(S)$, si du moins, ce que nous supposons, on emploie une méthode itérative donnant une bonne estimation des multiplicateurs. Soit φ^ν, la borne inférieure attachée à x^ν. Elle a la propriété :

$$\varphi^\nu \uparrow \hat{\varphi}_S, \qquad \text{lorsque} \qquad \nu \to +\infty.$$

Si donc on est dans le cas où $\hat{\varphi}_S > \hat{\varphi}$ (S à refuser), les calculs seront en fait arrêtés dès que $\varphi^\nu > \hat{\varphi}$.

Cas des variables bivalentes

Certaines des variables entières peuvent être des variables bivalentes : nous supposerons que les deux valeurs qu'elles peuvent prendre sont 0 et 1. Ce cas entre bien entendu dans le cadre général de la méthode, avec $a_j = 0$ et $b_j = 1$ pour toute variable x_j bivalente. Il est à remarquer que chacune des ailes correspondant à une variable bivalente se réduit à son origine.

Le cas le plus intéressant est celui où les N variables entières sont toutes bivalentes ; une simplification certaine de la programmation-machine en résulte puisque, pour chaque sommet, il ne peut y avoir que deux successeurs possibles, qui sont d'étage supérieur. Un gain important sur le nombre des mémoires est également possible puisque N *bits* suffisent pour mémoriser N variables bivalentes, permettant ainsi le traitement

rapide de très gros problèmes de ce type. Remarquons que le cas général se ramène à celui-ci, par développement de chaque variable entière dans le système de base 2 :

$$x_j = a_j + \sum_{k=0}^{n_j} 2^k x_{jk},\ 0 \leqslant x_{kj} \leqslant 1,$$

où n_j est déterminé par b_j. Nous ne discuterons pas ici les avantages et les inconvénients d'une telle transformation.

4. EXEMPLE ILLUSTRATIF

On traite complètement ci-après le problème P suivant :

P : minimiser

$$(5x_1 - 13)^2 + 15(130x_1 - 100x_2 - 23)^2 + 30(20x_1 + 11x_2 - 20x_3 - 10)^2$$

sous les contraintes :

$$0 \leqslant x_j \leqslant 6,\ j \in J = \{1, 2, 3\} \tag{1}$$

$$x_j \text{ entier},\ j \in E = J. \tag{3}$$

(l'ensemble convexe K est ici l'espace R^3 tout entier). La fonction φ est convexe et différentiable.

Si les seuls problèmes à résoudre étaient aussi « petits », la solution raisonnable, du moins pour une machine (ce n'est pas vrai pour l'homme !) serait de procéder à une énumération complète, ce qui exige le calcul de $\varphi(x)$ pour $7^3 = 343$ points.

La solution du problème $P(\varnothing)$ obtenue en négligeant (3), c'est-à-dire en traitant les variables comme variables réelles et non comme variables entières, est :

$$\hat{x}(\varnothing) = (2.6\ ;\ 3.15\ ;\ 3.7575),\ \varphi(\hat{x}(\varnothing)) = 0.$$

La solution de P est :

$$\hat{x} = (1\ ;\ 1\ ;\ 1),\ \varphi(\hat{x}) = 829.$$

La solution obtenue par le « meilleur arrondi » est :

$$x^0 = (3\ ;\ 3\ ;\ 4),\ \varphi(x^0) = 67\,609.$$

Parmi les huit points obtenus en arrondissant de toutes les façons $\hat{x}_j(\varnothing)$, $j = 1, 2, 3$, le meilleur est :

$$x^1 = (3\ ;\ 4\ ;\ 4),\ \varphi(x^1) = 21\,219.$$

Ainsi, le « meilleur arrondi » ne donne rien de bon (67 609). Le « meilleur de tous les arrondis possibles », bien qu'évidemment meilleur (21 219),

n'est encore guère recommandable. L'arrêt de l'algorithme à la première solution rencontrée donne déjà un résultat un peu meilleur (17 419, à la troisième itération), et l'arrêt au deuxième point entier rencontré donne une approximation (934, à la dixième itération) qui a le mérite d'être du même ordre de grandeur que la valeur optimale (829, quatorzième itération). Trois itérations supplémentaires sont nécessaires pour pouvoir affirmer, une fois la solution optimale obtenue, qu'elle a effectivement la propriété d'optimalité requise pour P.

Les calculs sont présentés sous forme d'un tableau à dix colonnes.

Colonne 1 : nom du sommet $T = (B, \bar{x}_B)$ examiné dans l'arborescence (0, 1, 2, ...) ; ce nom coïncide avec le numéro de l'itération.

Colonne 2 : ensemble B ; pour simplifier, B est toujours l'un des quatre suivants : $\varnothing$, $\{1\}$, $\{1, 2\}$, $\{1, 2, 3\}$; d'autres règles sont évidemment possibles : après avoir fixé x_1 à une valeur entière $\bar{x}_1$, on peut, au lieu de fixer x_2, choisir de fixer celle des deux inconnues x_2, x_3 qui est la plus éloignée (ou la plus proche) d'un entier. Tout cela, est un accord avec une certaine liberté laissée dans la règle R 2.

Colonne 3 : la notation $i \,.\, g$ signifie que le sommet étudié est le successeur du sommet i dans son aile, qui est une aile gauche ; la notation $i \,.\, sg$ signifie que le sommet étudié est le successeur d'étage supérieur de i, et qu'il débute une aile gauche ($i \,.\, d$ et $i \,.\, sd$ ont les mêmes significations, au remplacement près de « gauche » par « droite »).

Colonnes 4, 5, 6 : valeurs des composantes x_1, x_2, x_3 pour la solution optimale de $P(T)$; lorsqu'une valeur de x_j est soulignée, cela signifie qu'elle a été fixée à une valeur entière parce que $j \in A$; lorsqu'en outre elle est accompagnée d'un astérisque, c'est que $j = \beta$; $x_j = 6^o$ ne sert qu'à insister sur le fait que, x_j étant égal à 6, la contrainte $x_j \leqslant 6$ est *effective* (le symbole (^{o}) est mis pour « obstacle » ; même interprétation pour $x_j = 0^o$, avec la borne inférieure à zéro).

Colonnes 7, 8 : $\hat{\varphi}_T$, $\tilde{\varphi}_T$ comme ils ont été définis ci-dessus ; le meilleur point entier correspond au nombre souligné dans la colonne 7.

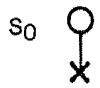
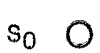
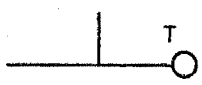
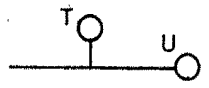
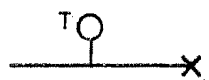
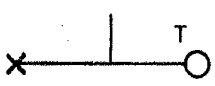
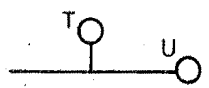
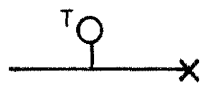
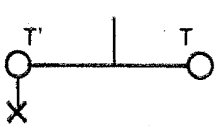
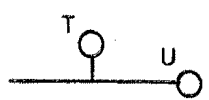
Colonne 9 : état de l'arborescence à la fin de l'itération en cours (le nom de la racine de l'arborescence est omis ; lorsque l'état de l'étage le plus élevé est l'état 9, on va directement en l'un des états 0, 6, 7, 8 qu'il engendre).

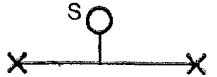
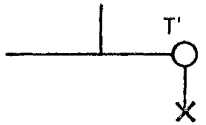
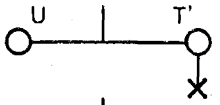
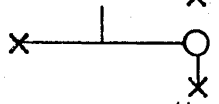
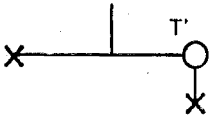
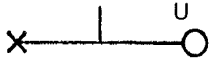
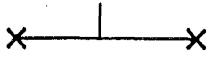
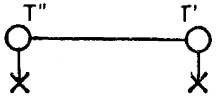
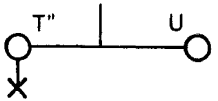
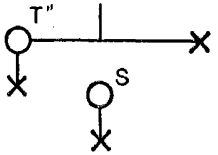
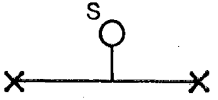
Colonne 10 : valeur de l'étage le plus élevé, t_m.

Les valeurs de $\hat{x}_j(T)$, $\hat{\varphi}_T$, $\tilde{\varphi}_T$ sont des valeurs exactes excepté pour $T = 9$ où des valeurs sont indiquées par excès ($^+$) ou par défaut ($^-$).

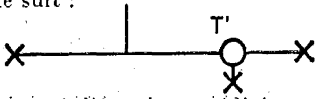
L'algorithme exige, après la minimisation à trois variables, 17 itérations qui se décomposent en : 5 minimisations à deux variables, 8 minimisations à une variable, plus 4 calculs de $\varphi(x)$.

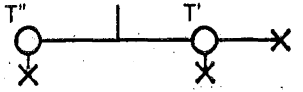
L'utilisation de procédures heuristiques est souvent recommandable, en conjonction avec la présente méthode, pourvu qu'elles soient simples et conduisent à des calculs peu nombreux. La connaissance d'une solution $\check{x}$

Avant				Apres		
N°	Etat	Etage	Règle	N°	Etat	Etage
0		0	R 1		FIN	
1		0	R 2 Acc.	2		1
			R 2 Ref.	5		1
2		t_m	R 2 Acc.	2		$t_m := t_m + 1$
			R 2 Ref.	5		$t_m := t_m + 1$
3		t_m	R 2 Acc.	2		$t_m := t_m + 1$
			R 2 Ref.	5		$t_m := t_m + 1$
4		t_m	R 2 Acc.	2		$t_m := t_m + 1$
			R 2 Ref.	5		$t_m := t_m + 1$
5		t_m	R 3 Acc.	3		$t_m := t_m$

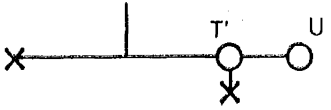
			R 3 Ref.	9		$t_m := t_m$
6		t_m	R 3 Acc.	4		$t_m := t_m$
			R 3 Ref.	7		$t_m := t_m$
7		t_m	R 3 Acc.	3		$t_m := t_m$
			R 3 Ref.	9		$t_m := t_m$
8		t_m	R 3 Acc.	4		$t_m := t_m$
			R 3 Ref.	7		$t_m := t_m$
9		t_m	R 4	0		$t_m := t_m - 1$

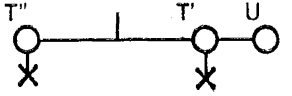
Commentaires T, T', T'' sont les sommets acceptés à l'étage le plus élevé $t_m > 0$ avant application des règles R 1 à R 4. Le sommet examiné, lorsqu'il y en a un, est U : son nom, n'est pas indiqué lorsqu'il est refusé, pour rappeler qu'on ne conserve pas en mémoire les sommets refusés. Dans les états 7 et 8, U est successeur de T' ; dans ces deux états, lorsque U est refusé, un état intermédiaire est schématisé comme suit :



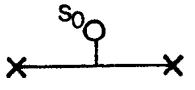
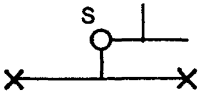
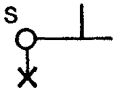
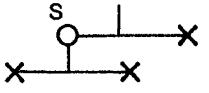
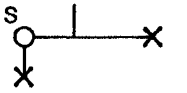
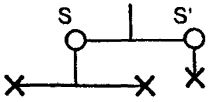
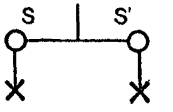


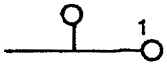
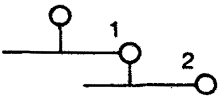
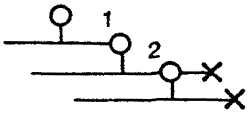
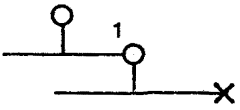
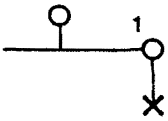
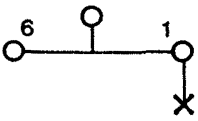
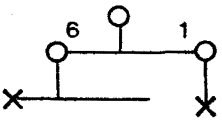
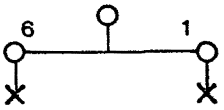
tandis que si U est accepté, l'état intermédiaire est :

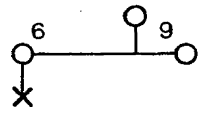
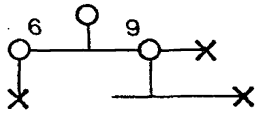
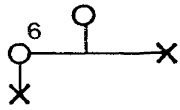
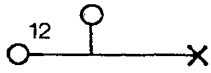
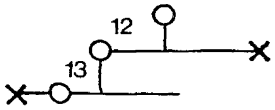
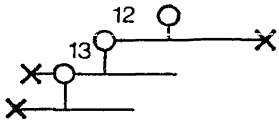
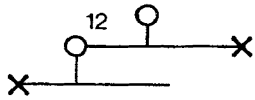
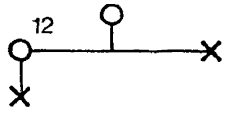




Il est transformé immédiatement dans les états finaux du tableau. S désigne, s'il y a lieu, le prédécesseur (direct) unique de l'étage t_m. La notation $t_m := t_m + \varepsilon$, $\varepsilon = 0$, 1 ou -1 signifie que l'étage le plus élevé avant l'application des règles R 1 à R 4 étant t_m, il est $t_m + \varepsilon$ après leur application. Dans l'état 9 pour t_m, on peut détailler les sous-cas, dépendant de l'état de l'étage $t_m - 1$ *avant* application des règles R 1 à R 4.

Avant				Après		
N°	Etat	Etage	Règle	N°	Etat	Etage
9.1		0 1	R 4	0		0
9.2		$t_m - 1$ t_m	R 4	6		$t_m := t_m - 1$
9.3		$t_m - 1$ t_m	R 4	7		$t_m := t_m - 1$
9.4		$t_m - 1$ t_m	R 4	8		$t_m := t_m - 1$

1	2	3	4	5	6	7	8	9	10
T	B	S	$\hat{x}(T)$ x_1	x_2	x_3	$\hat{\varphi}_T$	$\tilde{\varphi}_T$	Etat de l'arborescence	t_m
0	∅		2.6	3.15	3.7575	0			Début 0
1	1	0. *sd*	3*	3.67	4.5185	4	24		1
2	1,2	1. *sd*	3.	4*	4.7	16339	115339		2
3	1,2,3	2. *sd*	3.	4.	5*	17419			3
4	1,2,3	2. *sg*	3.	4.	4*	21219			2
5	1,2	1. *sg*	3.	3*	4.15	67339			1
6	1	0. *sg*	2*	2.37	2.8035	9	39		1
7	1,2	6. *sg*	2.	2*	2.06	20544			2
8	1,2	6. *sd*	2.	3*	3.15	59544			1

1	2	3	4	5	6	7	8	9	10
T	B	S	$\hat{x}(T)$ x_1	x_2	x_3	$\hat{\varphi}_T$	$\hat{\varphi}_T$	Etat de l'arborescence	t_m
9	1	1. *d*	4*	4.9600 +	6°	683−	10107−		1
10	1,2	9 . *sd*	4.	5*	6°	934			2
11	1,2	9. *sg*	4.	4*	5.7	141184			1
12	1	6. *g*	1*	1.07	1.0885	64	144		1
13	1,2	12. *sg*	1.	1*	1.05	799	21799		2
14	1,2,3	13. *sg*	1.	1.	1*	829			3
15	1,2,3	13. *sd*	1.	1.	2*	14029			2
16	1,2	12. *sd*	1.	2*	1.6	129735			1
17	1	12. *g*	0*	0°	0°	11104			0 FIN

Solution : $T = 14$

de (1), (2), (3) donne en effet, par résolution de $P(S)$ avec $S = (E, \overset{\vee}{x}_E)$, une valeur de $\hat{\varphi}$ que l'on peut espérer être assez petite pour permettre le refus d'un grand nombre de sommets de G, ce qui entraîne le non-examen d'un nombre de sommets d'autant plus grand que $\hat{\varphi}$ est plus petit. Le fait que la plupart des procédures heuristiques simples échouent dans l'exemple actuel n'est pas une contre-indication suffisante.

Appelons « Exemple 1 » l'exemple précédent, et « Exemple 2 » celui qui s'en déduit en supprimant les contraintes (2). Appelons M 1 la méthode exposée ci-dessus et M 2, la méthode suivante : à chaque itération on considère, parmi les sommets mémorisés, le sommet S pour lequel $\hat{\varphi}_S$ est le plus petit, et on examine les trois successeurs (deux d'étage supérieur, un dans l'aile de S — ce dernier n'existant pas si S est la racine — le successeur dans l'aile étant toujours le troisième) ; on applique le plus grand nombre possible des procédures de refus mentionnées au paragraphe 2, ainsi que, s'il y a lieu, la procédure de refus déduite de la convexité. On démontre comme ci-dessus que la méthode M 2 résoud aussi les problèmes quasi-convexes. Il s'agit d'ailleurs de la méthode de Land et Doig modifiée par les procédures de refus, et étendue à la programmation non-linéaire. Les calculs ne sont pas indiqués ici. Le lecteur intéressé pourra les obtenir auprès de l'auteur.

La comparaison portera sur les points suivants : nombre des minimisations à 1 variable (tableau 1), nombre des minimisations à 2 variables (tableau 2), nombre des sommets à mémoriser (tableau 3), numéro des itérations donnant des solutions entières (tableau 4, où les nombres entre parenthèses indiquent les valeurs correspondantes de φ).

	M 1	M 2
Exemple 1	8	8
Exemple 2	24	26

Tableau 1

	M 1	M 2
Exemple 1	5	6
Exemple 2	12	12

Tableau 2

	M 1	M 2
Exemple 1	3	8
Exemple 2	3	15

Tableau 3

	M 1	M 2
Exemple 1	3 (17 419), 10 (934), 14 (829)	11 (11 104), 12 (934), 15 (829)
Exemple 2	3 (17 419), 11 (934), 19 (829)	21 (934), 40 (829)

Tableau 4

En conclusion, il semble, au moins sur le vu de ces exemples, que l'on doive attendre de M 1, par rapport à M 2, les avantages suivants :

1. moins de minimisations (tableaux 1 et 2) ;
2. moins de sommets à mémoriser (tableau 3) ;
3. moins de temps pour obtenir les mêmes solutions entières (tableau 4) (avantage valable si l'on est amené à abandonner les calculs avant achèvement).

Il faut ajouter à cela l'avantage certain que le nombre des mémoires à réserver est au maximum $2N - 2 = 4$ sommets pour la méthode M 1. Outre que ce nombre est dépassé pour la méthode M 2 (tableau 3), il ne peut être borné *a priori*. Cela conduit, lorsqu'on cherche à appliquer M 2, soit à réserver N' mémoires centrales et à mémoriser sur les mémoires périphériques lorsque le nombre des sommets à conserver est supérieur à N (d'où difficultés de programmation et extrême lenteur), soit à oublier volontairement, dès qu'il y a $N' + 1$ sommets, le plus mauvais d'entre eux (au risque de ne pas atteindre l'optimum du problème posé).

5. METHODE ARBORESCENTE A N GROUPES DE MEMOIRES

Il se peut que mémoriser $2N - 2$ sommets soit encore trop pour certains gros problèmes. Nous allons indiquer comment conduire les opérations pour que le nombre maximum soit N au lieu de $2N - 2$. Il suffit de faire en sorte que chaque étage de valeur $t < t_m$ comporte un sommet accepté et un seul, tandis que l'étage t_m comporte 0 ou 1 sommet accepté (lorsque $t_m = N$, ce nombre sera 0). Il suffit de remplacer la règle R 3 par la règle R 3 *bis* suivante :

R 3 *bis*. Si l'étage t_m est dans l'état 6 ou 7, examiner le successeur de T' dans son aile et l'accepter, ou le refuser (ce qui amène à l'état 2,5 ou 9). Si l'étage t_m est dans l'état 5, examiner l'origine de l'aile opposée, et l'accepter ou la refuser (ce qui amène à l'état 3 ou 9).

On montre successivement chacun des résultats suivants :

Théorème 6. Dans l'application de l'algorithme, l'étage le plus élevé est, au début de toute itération, dans un des états 0, 1, 2, 3, 5, 6, 7, 9, et les étages moins élevés dans un des états 0, 1, 2, 3.

Théorème 7. A toute itération, l'état de l'arborescence ne comporte pas plus de N sommets acceptés.

Théorème 8. En un nombre fini d'itérations, l'algorithme modifié trouve une solution de P, ou bien conclut que les contraintes de P sont incompatibles.

Il est intéressant de voir ce que donne l'algorithme modifié sur l'exemple numérique du paragraphe précédent. On trouve exactement les

mêmes sommets, dans un ordre différent. En nommant les sommets comme ils l'ont été, cet ordre est :

0, 1, 2, *3*, 4, 5, 9, *10*, 11, 6, 7, 8, 12, 13, *14*, 15, 16, 17

(les sommets 10, 14 correspondent à des solutions entières).

La méthode modifiée paraît donc intéressante. Nous ne sommes cependant pas en mesure de la comparer, à d'autres points de vue que celui du nombre des mémoires, à la méthode exposée au paragraphe 2. Elle paraît posséder l'inconvénient *a priori* de trop explorer une aile, avant que la décision ne soit prise d'entamer l'examen de l'aile opposée.

6. METHODES A UN OU DEUX GROUPES DE COMPOSANTES

Dans tout ce qui précède, nous avons supposé qu'avec chaque sommet S sont mémorisées, outre un certain nombre d'informations relatives à l'arborescence, les composantes de la solution $\hat{x}(S)$ de $P(S)$. Bien entendu, cela sous-entend que, lorsque l'on passe d'un sommet S à un successeur d'étage supérieur, les composantes entières dont les indices appartiennent à A ne sont pas répétés effectivement. La mémorisation des composantes de $\hat{x}(S)$ est utile pour pouvoir démarrer en un point suffisamment bon les itérations de l'optimisation du problème correspondant à un successeur de S. Il faut noter ici que nos règles de refus peuvent amener à passer d'un étage t_m à l'étage $t_m - 100$, par exemple à l'itération suivante.

Lorsqu'il est tout à fait indispensable d'économiser au maximum les mémoires, on peut se contenter de la méthode à N sommets, et ne mémoriser que l'unique solution de l'étage le plus élevé. A cela doit naturellement s'ajouter la mémorisation de la meilleure solution rencontrée ayant des composantes x_j entières, $\forall\, j \in E$.

Lorsque l'étage t_m augmente ou reste le même, on procède d'une façon analogue à celle qui a été indiquée à la fin du paragraphe 2. Lorsque t_m diminue, on libère un certain nombre de composantes fixées à des nombres entiers, ce qui fournit immédiatement un point de départ pour l'optimisation « continue ».

Les remarques qui précèdent valent pour les *composantes* des solutions correspondant aux sommets de l'arborescence. Il est bien entendu nécessaire de conserver un petit nombre d'informations sur chacun de ces sommets, pour pouvoir continuer à engendrer l'arborescence et appliquer les règles de refus.

Il serait possible, en fait, de ne conserver en mémoire *aucune* des solutions correspondant aux sommets de l'arborescence, puisqu'on peut toujours démarrer les itérations du problème continu $P(S)$ avec : $x_j^0 = \bar{x}_j$, $\forall\, j \in A$; $x_j^0 = a_j$, $\forall\, j \notin A$. Outre qu'il faut de toute façon mémoriser $\bar{x}_j$,

$\forall j \in A$ (où A peut être E à une unité près), les temps de calcul seraient exagérément allongés.

On peut procéder de même avec la méthode à $2N - 1$ sommets, en mémorisant *une* solution correspondant à l'étage le plus élevé, ou deux si l'on est suffisamment à l'aise. Dans ce dernier cas, il apparaît bon de conserver systématiquement en mémoire la solution d'étage le plus élevé possible et appartenant à une aile droite, ainsi que la solution d'étage le plus élevé possible appartenant à une aile gauche (les deux étages n'étant pas nécessairement les mêmes).

Le procédé le plus efficace est sans doute le suivant : la machine calcule le nombre de mémoires attribuables à la conservation des solutions, décide du nombre de solutions à conserver, et répartit automatiquement ces solutions entre les étages les plus élevés et les autres.

CONCLUSION

Nous espérons avoir montré comment, étant donné un programme non-linéaire (quasi-convexe) partiellement ou totalement discret, on peut profiter au mieux des mémoires disponibles : une fois connu le nombre des mémoires non-utilisées par le programme et les différentes servitudes, la machine peut choisir elle-même une méthode à m groupes de mémoires, où m est le maximum compatible avec $1 \leqslant m \leqslant 2N - 2$ (le groupe nécessaire à la mémorisation de la meilleure solution entière rencontrée étant compté dans les servitudes).

REFERENCES

ABADIE (J.), ed., Nonlinear Programming, North Holland Publishing Company (1967).

ABADIE (J.), ed., Integer and Nonlinear Programming, North Holland Publishing Company (1970).

ABADIE (J.) et CARPENTIER (J.), Généralisation de la méthode du Gradient Réduit de Wolfe au cas des contraintes non-linéaires, pages 1047-1053 de *Proceedings of the Fourth IFORS Conference*, D. B. Hertz and J. Mélèse (ed.), Wiley — Interscience (1966).

ABADIE (J.) and CARPENTIER (J.), *Generalization of the Wolfe Reduced Gradient method to the case of nonlinear contraints*, in : Optimization, R. Fletcher (ed.), Academic Press (1969).

ABADIE (J.) et GUIGOU (J.), *Gradient Réduit Généralisé*, Note E.D.F. HI 069/02 (1969).

ABADIE (J.) and GUIGOU (J). *Numerical experiments with the GRG method*, in Abadie (1970).

BALINSKI (M. L.), *Some general methods in integer programming*, Chap. IX of Abadie, 1967 (1967).

BERGE (C.), *Théorie des Graphes et ses applications*, Dunod (1958).

CARPENTIER (J.), *Generalized Reduced Gradient algorithm, combination with integer programming, application to an electric power system problem (mimeographed)*, paper presentet to the Nato Advanced Study Institute on Integer and Nonlinear Programming, Bandol, 8-10 june 1969 (1969).

CARPENTIER (J.), CASSAPOGLOU (C.) et al., *Une méthode générale de résolution des problèmes de dispatching économique avec variables entières, tenant compte des coûts de marche à vide des groupes*, paper presented at the International Conference on Operational Research & Power Systems, Athens (1968).

COLVILLE (A. R.), *A comparative study of nonlinear programming codes*, IBM NYSC Report 320-2949 (1968).

FLETCHER (R.), éd., *Optimization*, Academic Press (1969).

HERVE (P.), *Les procédures arborescentes d'optimisation*, R.I.R.O., *14*, 69-80 (1968).

LAND (A. H.) and DOIG (A. G.), *An automatic method of solving discrete programming problems*, Econometrica, *28*, 497-520 (1960).

ROY (B.), *Algèbre moderne et théorie des graphes*, Dunod (1969).

ROY (B)., *Procédure d'exploration par séparation et évaluation (P.S.E.P. et P.S.E.S.)*, R.I.R.O., 3e année, nº V-1, 61-90 (1969).

ROY (B.), BENAYOUN (R.) and TERGNY (J.), *From S.E.P. procedure to the mixed Ophélie programme, in* Abadie (1970).