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

JACQUES DELORME
EDITH HEURGON

Problèmes de partitionnement : exploration arborescente ou méthode de troncatures ?

Revue française d'automatique, informatique, recherche opérationnelle. Recherche opérationnelle, tome 9, n° V2 (1975), p. 53-65

<http://www.numdam.org/item?id=RO_1975__9_2_53_0>

R.A.I.R.O.
(9e année, juin 1975, V-2, p. 53 à 65)

PROBLEMES DE PARTITIONNEMENT : EXPLORATION ARBORESCENTE OU METHODE DE TRONCATURES ? (*)

par Jacques DELORME (1) et Edith HEURGON (1)

Résumé. — *Cet article fait le point sur les méthodes de résolution des problèmes de recouvrement exact (avec quelques contraintes supplémentaires), appliquées de façon opérationnelle à la R.A.T.P. Après une première phase de programmation linéaire, la première consiste en une exploration arborescente, la seconde applique une méthode de troncatures. Pour des problèmes comportant de 1 000 à 2 500 variables et de 50 à 150 contraintes, les résultats sont très satisfaisants. Une des conclusions de ce travail est que, contrairement à ce que l'on croit souvent, les méthodes de troncatures sont particulièrement efficaces pour résoudre les problèmes de partitionnement de grande taille.*

Un premier article [9], publié en 1972, analysait les méthodes mises au point à la R.A.T.P. afin d'automatiser l'habillage des horaires des lignes d'autobus. A l'époque, les conclusions laissaient entendre que, si 90 % des cas étaient résolus d'une manière satisfaisante par ordinateur, environ 10 % des problèmes ne pouvaient encore être traités (soit en raison du temps excessif de calcul, soit par absence de solution réalisable). Après plusieurs années d'étude, ces résultats ont été très largement améliorés, tant au niveau de la qualité des services obtenus, qu'à celui du pourcentage des problèmes résolus automatiquement et des temps de calcul sur ordinateur.

Nous rappellerons brièvement le problème posé et sa formulation mathématique. Puis, après avoir évoqué les travaux relatifs à la mise au point d'un bon générateur de services, nous insisterons plus particulièrement sur les méthodes de résolution développées récemment. Après une première phase de programmation linéaire, l'une consiste en un algorithme d'énumération implicite, inspiré de celui de Garfinkel et Nemhauser [3], l'autre en une méthode de troncatures, dite méthode des congruences décroissantes, due à Michel Gon-

(*) Reçu octobre 1974.

(1) Ingénieurs chargés de recherche au service de la Recherche opérationnelle et du Calcul économique de la R.A.T.P.

dran [5]. Les deux chaînes de programmes qui en résultent exigent le plus souvent des temps d'exécution comparables et sont utilisées dans la phase actuelle d'une manière complémentaire, l'une pouvant résoudre les problèmes non traités par l'autre. Dans l'avenir cependant, l'application de la méthode de troncatures semble devoir se généraliser : souvent plus rapide, elle présente l'indéniable avantage de fournir toujours la solution entière optimale. Elle permet aussi la prise en compte aisée de contraintes additionnelles et de toute modification dans la formulation du programme. En outre, pour les problèmes de plus grande taille, elle manifeste une très nette supériorité.

I. LE PROBLÈME. SA FORMULATION MATHÉMATIQUE

I.1. Définition

L'habillage des horaires d'une ligne d'autobus consiste à découper un horaire, préalablement établi, en un nombre fixé de postes (ou services), respectant les conditions de travail du personnel.

I.2. Formulation

Si l'on suppose la génération préalable d'un bon échantillon de services possibles ($j, j \in J$), non nécessairement compatibles entre eux, chacun muni d'un coût c_j, le problème revient à sélectionner N éléments dans cet ensemble (N étant donné) assurant la couverture exacte de l'horaire et minimisant la fonction économique. Il s'agit là d'un problème classique de partitionnement dont la formulation est :

$$(P)\quad \begin{cases} \min \sum_{j \in J} c_j x_j & (1) \\ \sum_{j \in J} a_{ij} x_j = 1 \qquad \forall i \in I & (2) \\ \sum_{j \in J} x_j = N & (3) \\ x_j = 0, 1 \qquad \forall j \in J\,. & (4) \end{cases}$$

Le problème spécifique de la R.A.T.P. exige l'adjonction d'une contrainte supplémentaire de type quelconque :

$$\sum_{j \in J} t_j x_j \leqslant T \qquad (5)$$

avec c_j, $t_j \geqslant 0$, $a_{ij} = 0$ ou 1, N et T, c_j et $t_j \in N$.

I.3. Dimensions

S'agissant d'un problème extrêmement combinatoire, une des principales difficultés a été la mise au point d'*un générateur de services adéquat et souple* ;

adéquat, c'est-à-dire produisant un ensemble de services qui, sans être trop important, soit néanmoins porteur de solutions satisfaisantes ; souple, car, en dépit de l'extrême complexité des conditions de travail, il doit s'adapter promptement à toutes modifications de celles-ci. Pour limiter la taille du problème, le générateur a été enrichi d'un prégénérateur. Ce programme tente, dans une large mesure, de prendre en compte l'expérience des spécialistes des horaires et notamment de fixer a priori certains services.

Aussi, pour l'habillage des lignes d'autobus, le problème (P) comporte entre 1 000 et 2 500 variables et de 80 à 150 contraintes. Signalons que des études parallèles sont menées actuellement pour adapter ces méthodes à l'habillage des lignes de métro. Ce sont alors des problèmes d'une toute autre ampleur (de 600 à 800 contraintes, sans décomposition de l'horaire) qui doivent être considérés.

I.4. Résolution

La résolution du problème (P), augmenté de la contrainte (5), s'effectue en deux phases :

1) Résolution du programme linéaire continu. Si la solution optimale est entière, le problème est résolu. Notons que la présence de la contrainte (5) réduit de façon très importante le pourcentage des cas où cette condition est vérifiée (il passe de 70 % à 30 %).

2) Résolution du programme en nombres entiers,

— soit par énumération implicite,

— soit par introduction de troncatures.

Après avoir développé les méthodes relatives à cette deuxième phase, nous reviendrons brièvement sur la première : la programmation linéaire continue.

II. ALGORITHME D'ÉNUMÉRATION IMPLICITE

La méthode finalement adoptée se distingue de l'algorithme de Garfinkel et Nemhauser, sur lequel nous ne reviendrons pas ici [3], par les *critères de choix utilisés* dans l'élaboration des listes hiérarchiques et le *calcul de la fonction d'évaluation*.

La nature du problème à résoudre (extrêmement combinatoire, mais aussi très contraint avec notamment l'égalité (3)) rend l'application du seul algorithme d'énumération implicite inefficace car trop énumératif. Aussi, la première phase (programmation linéaire continue) a-t-elle été jugée indispensable. Nous avons donc fait en sorte *d'utiliser au maximum les résultats fournis par la solution continue* du programme linéaire, notamment *les variables* directement *arbitrées à* 1 *et les coûts marginaux* de celles-ci.

1) Ainsi la *solution partielle* est-elle au départ *initialisée par les variables fixées à* 1 dans la solution continue. A la différence des méthodes employées à l'époque par Air France, ce processus *ne consiste pas à fixer définitivement* les valeurs de ces variables, mais seulement à orienter le cheminement dans l'arborescence étudiée. Ces variables, dans la suite de l'algorithme, pourront, comme toute autre, être à l'origine de retours en arrière. Signalons néanmoins que, dans les expériences réalisées, rares sont les fois où ces premiers choix ont été remis en cause.

2) *L'utilisation des coûts marginaux* permet d'accélérer considérablement la procédure de recherche arborescente. Pour toute solution de (P), la valeur de la fonction économique est donnée par :

$$f = f^0 + \sum_{j \in J} d_j x_j = \sum_{j \in J} c_j x_j \ (^1)$$

où d_j désigne le coût marginal de la variable x_j.

— *Les coûts marginaux* conduisent à *une fonction de choix beaucoup plus fine* que les coûts réels (pour une égalité donnée, cette fonction de choix établit un ordre entre les variables pouvant la vérifier : c'est donc selon leur coût marginal non décroissant que les variables sont classées à l'intérieur des listes hiérarchiques). *On peut ainsi limiter de façon remarquable les remontées dans l'arborescence.*

— D'autre part, dans la méthode d'énumération implicite proposée par Garfinkel et Nemhauser, la fonction d'évaluation n'est autre que le coût de la solution partielle obtenue. Dans la mesure où, dans cette catégorie de problèmes, optimum continu et optimum entier sont très voisins, *l'évaluation d'une solution partielle* par la formule :

$$f_0 + \sum_{(x_j = 1)} d_j$$

atteint rapidement le coût de la dernière solution trouvée. Le processus d'optimisation exige alors beaucoup moins de temps, la *fonction d'évaluation se trouvant ainsi nettement améliorée.*

— *Réduction du problème*

Naturellement, ce sont les variables de faible coût marginal qui altèrent le moins la valeur de la fonction économique. Si donc le problème initial comporte n variables, on peut essayer de chercher une solution dans le sous-ensemble des n_1 variables de plus petit coût marginal. (On choisit généralement n_1 multiple du nombre des contraintes.) Les dimensions du sous-problème (P_1) correspondant étant assez restreintes, il est souvent possible de

(1) Cette formule peut être adaptée facilement dans le cas où l'inégalité (5) doit être prise en compte.

démontrer, dans ce sous-problème, l'optimalité d'une solution X_1. Que vaut cette solution pour le problème initial (P) ? Naturellement, elle constitue une solution de P.

Soit d_{n_1+1} le plus petit coût marginal des variables contenues dans $P - P_1$: si le coût f_1 de la solution optimale X_1 de (P_1) est tel que :

$$f_1 \leqslant f^0 + d_{n_1+1} ,$$

il est clair que X_1 est solution optimale de (P). Sinon, on ne peut rien dire.

— *Le calcul est arrêté* dès que l'on a obtenu une solution réalisable à moins de t % de f^0, valeur de la fonction économique à l'optimum continu. En effet, si l'on trouve le plus souvent très vite des solutions réalisables, la vérification de l'optimalité exige un temps trop important. Ainsi, avons-nous convenu d'arrêter le déroulement de l'algorithme dès qu'une « bonne solution » était atteinte.

III. MÉTHODE DE TRONCATURES

Optimale, entière, réalisable, telles sont les propriétés que doit respecter la solution du programme linéaire en nombres entiers (P) que nous cherchons à déterminer. Par l'introduction de contraintes supplémentaires, les méthodes de troncatures se proposent de tronquer l'espace des solutions sans en soustraire aucune qui soit dotée à la fois des trois précédentes caractéristiques.

Le fait mathématique remarquable est qu'on peut engendrer systématiquement de telles contraintes de manière à ce que, en un nombre fini d'étapes, on obtienne soit la solution optimale, soit la preuve qu'il n'en existe aucune.

Chaque itération d'une méthode de troncatures admet deux phases :

a) par un algorithme de programmation linéaire classique, on détermine une solution de base (qui peut être, par exemple, primale admissible, duale admissible, optimale, entière, etc.) ;

b) si cette solution ne possède pas les trois propriétés précitées, on engendre une (ou plusieurs) nouvelle(s) troncature(s) que l'on introduit alors dans le précédent tableau simplicial.

Ce sont donc d'une part les caractéristiques de la solution de base choisie, d'autre part le mode de calcul des contraintes qui distinguent les différentes méthodes de troncatures.

Dans la méthode utilisée à la R.A.T.P. — dite méthode des congruences décroissantes [5] — la solution de base choisie à la phase *a*) est optimale et réalisable. Dès qu'une solution entière est obtenue, le processus itératif cesse.

III.1. Calcul d'une troncature

A l'optimum continu d'un programme linéaire (P_0), les variables de base et la fonction économique s'écrivent en fonction des variables hors base :

$$x_i = \bar{x}_i - \sum_{j \in N} \alpha_{ij} x_j \qquad \forall i \in B \tag{1}$$

où B et N désignent respectivement les ensembles des indices des variables de base et des variables hors-base, et $\bar{x}_i$ les seconds membres.

Si l'on suppose l'intégrité des variables x_i, il est aisé de vérifier que, pour un i donné (tel que $\bar{x}_i$ soit non entier), la relation (1) entraîne l'équation de congruence :

$$\sum_{j \in N} \alpha'_{ij} x_j \equiv \bar{x}'_i \qquad (\text{mod } D)$$

avec

$$\alpha'_{ij} = D\alpha_{ij} \text{ entier}$$
$$\bar{x}'_i = D\bar{x}_i \text{ entier},$$

et D déterminant de la base B.

Notons alors respectivement F_j et F_0 les représentants canoniques des classes d'équivalence modulo D, de α_{ij} et de $\bar{x}'_i$.

Il vient :

$$\sum_{j \in N} F_j x_j \equiv F_0 \qquad (\text{mod } D)\,. \tag{2}$$

Le membre de gauche de (2), représentant de la classe d'équivalence de F_0, est positif ou nul, puisque les coefficients F_j le sont. Comme, par définition, F_0 est le plus petit représentant, on peut écrire :

$$\sum_{j \in N} F_j x_j = F_0 + \lambda D \qquad \lambda \in N\,.$$

Si l'on divise par D, on obtient la troncature de Gomory I [4] :

$$\sum_{j \in N} f_j x_j \geqslant f_0\,.$$

Il s'agit bien d'une troncature car, d'une part, on n'élimine aucun point entier (seule l'intégrité des variables a été supposée), d'autre part l'optimum continu n'est plus solution réalisable (pour $x_N = 0$, on a :

$$\sum_{j \in N} f_j x_j = 0 < f_0\Big)\,.$$

III.2. Choix d'une troncature

Bien souvent, l'addition d'une seule troncature est insuffisante pour atteindre la solution optimale du problème et c'est plusieurs contraintes qu'il

faut introduire simultanément ou successivement. Se pose alors la question du critère de choix de la (ou des) contraintes additionnelles. En effet, à une itération donnée, l'ensemble des troncatures susceptibles d'être ajoutées au tableau simplicial constitue un groupe abélien fini (cyclique ou non).

1) *Plus grand second membre*

Si l'on multiplie les deux membres de la relation de congruence (2) par un nombre entier h (sans changer le modulo), on a :

$$\sum_{j \in N} hF_j x_j \equiv hF_0 \qquad (\text{mod } D) .$$

Soient respectivement F_j^h et F_0^h les représentants canoniques des classes d'équivalence de hF_j et hF_0. En divisant par D, on obtient une autre troncature :

$$\sum_{j \in N} f_j^h x_j \geqslant f_0^h \qquad \text{si} \qquad f_0^h \neq 0 .$$

Intuitivement, il semble que plus f_0^h est grand, plus est « profonde » la troncature. Toutefois, la multiplication par h transforme les coefficients de la relation, donc aussi « l'orientation de la troncature ». Alors que choisir ? On pourrait imaginer de prendre la contrainte qui assure la plus grande variation de la fonction économique. Mais, à moins de calculer cette variation pour chaque troncature, on n'en connaît pas la valeur. Une bonne heuristique consiste, semble-t-il, à choisir f_0^h le plus grand possible.

On montre que, si d désigne le PGCD de F_0 et de D, on peut trouver un entier h tel que

$$F_0^h = D - d ,$$

la valeur maximale qu'on puisse atteindre.

2) *Introduction de plusieurs troncatures à la fois*

Introduisons la troncature :

$$\sum_{j \in N} - f_j x_j + s = - f_0 .$$

Puisque le second membre $- f_0$ de cette contrainte est négatif, la base $B \cup \{ s \}$ du nouveau problème n'est pas primale réalisable. En revanche, comme les coûts marginaux demeurent inchangés, $B \cup \{ s \}$ est duale réalisable : on effectue alors l'optimisation par la méthode duale du simplexe. Tous les seconds membres étant positifs, sauf celui de la contrainte additionnelle, le pivotage s'effectue sur un élément f_{j^*} de cette troncature.

$$f_{j^*} \text{ est de la forme } \frac{F_{j^*}}{D} \begin{cases} F_j \in N \\ 0 < F_j < D \end{cases}$$

si D est le déterminant d'une base optimale B de (P_0).

Après pivotage, le déterminant a pour valeur :

$$D' = D.f_{j^*} = D \cdot \frac{F_{j^*}}{D} = F_{j^*} < D .$$

Si le second membre n'est pas entier, on peut engendrer une autre troncature et effectuer un nouveau pivotage. Comme le déterminant décroît strictement, on obtient une solution entière en un nombre fini d'opérations de ce type.

Remarquons cependant que, si le second membre devient entier, tous ses éléments ne sont pas forcément positifs ou nuls. Dans un tel cas, il faut calculer l'optimum du dual afin de les rendre non négatifs. Après cette optimisation, le second membre peut n'être plus entier. Il convient alors de poursuivre le processus en engendrant une nouvelle suite de troncatures (cf. figure 1 ci-dessous).

C'est seulement lorsque les éléments du second membre sont *non négatifs et entiers* que l'algorithme s'achève. On dit qu'on introduit « plusieurs troncatures à la fois » dans la mesure où l'on ne calcule pas l'optimum du dual pour chacune d'elles.

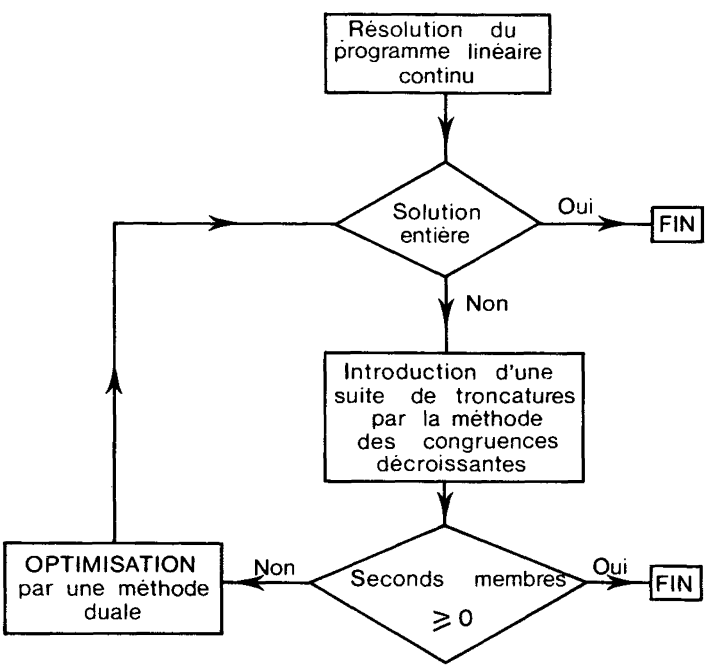

Figure 1

IV. PROGRAMMATION LINÉAIRE CONTINUE

Bien que les grands constructeurs d'ordinateurs proposent en général un code linéaire, il nous a paru nécessaire d'en écrire un pour plusieurs raisons.

— Le premier avantage est un gain de temps de calcul. En effet, les codes proposés par les constructeurs sont capables de résoudre des problèmes très différents, tant au point de vue de leur taille que de leur structure. Ils offrent fréquemment de nombreuses options intéressantes, mais elles l'alourdissent, et sont inutiles pour la résolution de notre problème.

D'autre part, une matrice faiblement remplie, dont les coefficients sont des 0 ou des 1 (exception faite des coûts, de la contrainte supplémentaire et du nombre d'équipes) possède une structure très particulière qui doit permettre des simplifications importantes, ainsi que de notables gains de temps et de mémoire.

— On gagne, par ailleurs, à ce que le code soit indépendant du matériel utilisé. En effet, les entrées-sorties varient avec les codes proposés, ce qui oblige, en cas de changement de matériel, à des modifications du programme qui génère les données et de celui qui utilise les résultats de la programmation linéaire.

— Enfin, un code général est une « boîte noire » et il est impossible d'en modifier aisément le contenu. Un code, écrit par nos soins, permet au contraire d'adapter le programme à toutes nos exigences et d'atteindre facilement les informations désirées. En particulier, les entrées-sorties peuvent être réduites au strict minimum, dans le format désiré, et par là, procurer un gain de temps appréciable dans la liaison avec les programmes situés en amont et en aval.

La méthode utilisée est appelée méthode des *multiplicateurs*. L'inverse de la base est calculé de façon explicite à chaque itération. Il suffit, en effet, de pouvoir calculer d'une part les coûts marginaux (pour choisir la variable entrant dans la base), d'autre part la colonne entrante et le second membre (pour déterminer la variable sortante). C'est pourquoi on applique la méthode usuelle de transformations successives de la matrice par pivotage à chaque itération, en se limitant à la seule sous-matrice carrée constituée au départ par les colonnes des variables artificielles.

Cette méthode est particulièrement intéressante lorsque la matrice initiale est faiblement remplie (ce qui est le cas de notre problème).

La mise au point d'un programme linéaire, destiné à résoudre des problèmes de taille importante, est une opération longue et laborieuse. De nombreuses difficultés ont été rencontrées au cours de son élaboration, posées notamment par l'élimination des variables artificielles, le traitement des dégénérescences et la précision. Ce travail trouve cependant sa justification dans le fait que les

temps d'exécution sur l'ordinateur de la Régie (Honeywell Bull 6050) sont réduits d'un tiers par rapport à ceux fournis par le code général L.P. 6000.

Méthode duale avec variables bornées

La formulation du problème que nous avons donnée est :

$$\text{Min } z = \sum_{j \in J} c_j x_j \tag{1}$$

$$\sum_{j \in J} a_{ij} x_j = 1 \qquad \forall i \in I \tag{2}$$

$$\sum_{j \in J} x_j = N \tag{3}$$

$$x_j = 0, 1 , \qquad \forall j \in J . \tag{4}$$

L'équation (3) est du même type que les équations (2) au second membre près. Transformons ces équations de la façon suivante :

$$\sum_{j \in J} a_{ij} x_j \leqslant 1 \qquad \forall i \in I$$

$$\sum_{j \in J} n_j x_j \geqslant m \tag{2'}$$

avec $n_j = \sum_i a_{ij}$: nombre de 1 dans la colonne j. Les équations (2') sont équivalentes aux équations (2). En introduisant des variables d'écart :

$$\sum_{j \in J} a_{ij} x_j + x_{n+i} = 1$$

$$\sum_{j \in J} - n_j x_j + x_{n+m+1} = - m .$$

on obtient une solution initiale, de base, duale-réalisable :

$$x_{n+i} = 1 \qquad \forall i \in (1, \ldots, m)$$

et

$$x_{n+m+1} = - m .$$

On utilise la méthode duale du simplexe pour calculer l'optimum de ce programme. On accélère l'algorithme en considérant que les variables sont bornées à 1. En effet, les équations (2) impliquent que, dans toute solution réalisable et positive, les variables sont inférieures ou égales à 1 :

$$\sum_{j \in J} a_{ij} x_j = 1$$

ou

$$\sum_{a_{ij} \neq 0} x_j = 1 \Rightarrow x_j \leqslant 1 \qquad \forall j \in J .$$

En prenant garde de ne pas faire entrer dans la base une variable qui prendrait une valeur supérieure à 1, on améliore considérablement les performances de l'algorithme dual (gains pouvant s'élever jusqu'à 75 % du temps de calcul). On atteint, en effet, plus rapidement une solution réalisable, donc optimale.

Dans ces conditions, les résultats obtenus avec l'algorithme dual sont en général meilleurs que ceux obtenus avec l'algorithme primal.

L'expérience a montré que :

— le temps de calcul est souvent plus court,

— le nombre d'itérations est inférieur,

— les dégénérescences sont moins nombreuses (il y a moins d'itérations donnant une variation de la fonction économique nulle),

— la valeur du plus grand déterminant rencontré en cours d'itérations est nettement plus faible (d'une centaine à quelques milliers), d'où une meilleure précision dans les calculs (réinversions de la base inutiles).

Les premiers résultats obtenus avec la méthode duale semblent prometteurs :

M x N	Primal	Dual
84 x 1427	5 mn 10s	3 mn 16s
69 x 1691	4 mn 50s	3 mn 05s
84 x 1112	4 mn 55s	2 mn 35s
102 x 1896	8 mn 10s	7 mn 20s

M : nombre de contraintes
N : nombre de variables

V. RÉSULTATS

V.1. Programmation linéaire et énumération implicite

Le tableau suivant présente un certain nombre de résultats. L'occupation de la mémoire est de 60 K mots de 36 bits sur l'ordinateur H.B. 6050.

N : nombre de variables,
M : nombre de contraintes,
T_{PL} : temps d'exécution du programme linéaire continu,
T_{PLE} : temps d'exécution du programme en entier,
$t\,\%$: pourcentage de la solution entière par rapport à l'optimum continu.

N	M	T_{PL}	T_{PLE}	t %
1043	69	2mn02s	1mn00s	2.9
948	78	3mn00s	0mn40s	0.5
1497	103	5mn06s	0mn24s	0
1850	111	9mn36s	1mn52s	0.3
2225	85	10mn48s	0mn40s	0.5
2275	92	7mn03s	2mn00s	0
2243	114	9mn36s	2mn09s	1.
2453	123	13mn48s	3mn12s	1.2

Ces résultats sont satisfaisants dans la mesure où les temps de calcul (pour la partie énumération implicite) sont relativement faibles et où la valeur de la solution entière est assez proche de l'optimum continu. Il est à remarquer que c'est la programmation linéaire qui exige le plus de temps-machine. Elle nous paraît néanmoins indispensable tant par les informations qu'elle fournit que par le fait qu'environ 30 % des solutions se trouvent directement entières. Dans le cas de solutions fractionnaires, certaines variables sont arbitrées à 1 et plus leur nombre est élevé, plus la recherche d'une solution entière est facilitée. Pour 2 % des problèmes traités, l'énumération implicite n'a donné aucune solution en moins de 6 mm CPU.

Bien que la contrainte supplémentaire altère les performances de l'algorithme de façon très sensible, l'utilisation des coûts marginaux, qui permet une exploration « intelligente » de l'arborescence, a rendu le programme plus efficace.

V.2. Programmation linéaire (méthode primale) et troncatures

Le tableau ci-dessous présente divers résultats. L'occupation de la mémoire est de 90 K mots de 36 bits sur l'ordinateur H.B. 6050.

N	M	T_{PL}	T_{PLE}	t %
953	88	3mn02s	2s	0,30
969	77	2mn51s	46s	1,
1019	91	4mn30s	28s	0,01
2116	115	5mn30s	10s	3,0
2186	111	8mn42s	2mn32s	0,25
1043	185	15mn00s	12s	0,6
1514	200	33mn29s	11s	0,03

Pour un certain nombre d'exemples, les temps d'exécution des deux programmes sont comparables. Comme ils utilisent tous les deux, dans un premier temps, la programmation linéaire continue, il est intéressant de remarquer qu'ils sont souvent complémentaires, l'un permettant de résoudre les problèmes sur lesquels l'autre a échoué (ainsi la très grande majorité des problèmes non résolus par l'algorithme énumératif l'ont été par la méthode de troncatures). Cette dernière présente divers avantages : elle fournit toujours la solution optimale du problème et prend aisément en compte les contraintes supplémentaires. En outre, elle permet seule de résoudre les problèmes qui dépassent 150 lignes.

VI. CONCLUSIONS

Les résultats précédents montrent donc clairement que, malgré diverses affirmations contraires, les problèmes de partitionnement constituent un champ d'action privilégié des méthodes de troncatures. Certes, pour l'instant, elles requièrent une phase préalable de programmation linéaire encore assez coûteuse. Seuls des efforts sur ce dernier point permettraient, semble-t-il, d'améliorer de manière significative les performances de ces programmes.

La mise en route opérationnelle de l'habillage automatique des horaires est réalisée progressivement à la R.A.T.P. (2 dépôts en 1973, 4 en 1974, etc.). Les problèmes qu'elle pose concernent maintenant l'organisation du travail, la formation du personnel, les relations avec d'autres applications de l'informatique, etc.

BIBLIOGRAPHIE

[1] DELORME J., Contribution à la résolution du problème de recouvrement : méthodes de troncature. *Thèse de Docteur-Ingénieur, Université de Paris VI*, juin 1974.

[2] DELORME J., HEURGON E., Set covering problems by linear programming and branch and bound algorithm, *VIII^e Symposium de programmation mathématique, Stanford* (août 1973).

[3] GARFINKEL R. S. and NEMHAUSER G. L., Set partitioning problem : set covering with equality constraints, *Operations Research*, 17 (1969), 848-856.

[4] GOMORY R. E., An algorithm for integer solutions to linear programs, *Princeton I.B.M. Math. Research Project*, technic report n° 1 (17 novembre 1958).

[5] GONDRAN M., Un outil pour la programmation en nombres entiers : la méthode des congruences décroissantes, *Revue d'Automatique, Informatique, Recherche Opérationnelle*, 7^e année (1973), vol. 3, 35-54.

[6] GONDRAN M., Problèmes combinatoires et programmation en nombres entiers, *Thèse d'Etat, Paris VI* (1974).

[7] GONDRAN M., An efficient cutting-plane algorithm by the method of decreasing congruences, *VIII^e Symposium de programmation mathématique*, Stanford (août 1973).

[8] GONDRAN M. et LAURIÈRE J. L., Un algorithme pour le problème de partitionnement, *Revue d'automatique, Informatique, Recherche Opérationnelle* 8^e année (1974), V.I., pp. 27-40.

[9] HEURGON E., Un problème de recouvrement : l'habillage des horaires d'une ligne d'autobus, *Revue Française d'Automatique, Informatique, Recherche Opérationnelle*, 6^e année, vol. 1 (1972).