Mathematical Programming 1 (1971) 76–94. North-Holland Publishing Company

EXPERIMENTS IN MIXED–INTEGER LINEAR PROGRAMMING *

M. BENICHOU, J.M. GAUTHIER, P. GIRODET, G. HENTGES
G. RIBIERE and O. VINCENT
IBM, France, Paris

Received 9 November 1970
Revised manuscript received 12 February 1971

This paper presents a "branch and bound" method for solving mixed integer linear programming problems. After briefly discussing the bases of the method, new concepts called pseudo-costs and estimations are introduced. Then, the heuristic rules for generating the tree, which are the main features of the method, are presented. Numerous parameters allow the user for adjusting the search strategy to a given problem.

This method has been implemented in the IBM Extended Mathematical Programming System in order to solve large mixed integer L. P. problems. Numerical results making comparisons between different choices of rules are provided and discussed.

1. Introduction

This paper presents a "branch and bound" method for solving mixed integer linear programming problems. After briefly discussing the bases of the method, new concepts called pseudo-costs and estimations are introduced. Then, the heuristic rules for generating the tree, which are the main features of the method, are presented. Numerous parameters allow the user to adjust the search strategy for a given problem.

This method has been implemented in the IBM Extended Mathematical Programming System in order to solve large mixed integer L. P. problems. An early version of this method, not including all the features described here, has been adapted in a code presently available in all IBM World Trade Data Centers. Numerical results making comparisons between different choices of rules are provided and discussed.

* This paper was presented at the 7th Mathematical Programming Symposium The Hague, The Netherlands.

2. "Branch and bound" method

2.1. *Problem Statement*

Any mixed integer linear program can be written in the following way:

Let X and Y denote two column vectors the components of which are x_i, $i = 1$ to p and y_j, $j = 1$ to q respectively. The two rectangular matrices A and B are of the order (m, p) and (m, q) respectively. Then the problem is:

Problem M

$$\min F = A_0^{\mathrm{T}} X + B_0^{\mathrm{T}} Y \tag{2.1}$$

under the constraints

$$AX + BY = D \tag{2.2}$$

$$\alpha_j \leqslant y_j \leqslant \beta_j \tag{2.3}$$

$$y_j \text{ integer} \qquad j = 1 \text{ to } q. \tag{2.4}$$

$$0 \leqslant x_i \qquad i = 1 \text{ to } p. \tag{2.5}$$

The x_i are the *continuous variables*, whereas the y_j are the *integer variables*. The bounds over the y_j must be finite, but can be either positive or negative. The problem, obtained when removing the integrality condition (2.4), is called the *continuous problem C*.

An *integer solution* of M or C is a set of values for the x_i and the y_j satisfying (2.2) to (2.5). An *optimal integer solution* is an integer solution minimizing F (in this paper, minimization will always be assumed).

2.2. *Stages of the Method*

The method involves two stages:

– First an *optimal continuous solution* of C is searched for by means of the usual linear programming methods. If this solution is integer, problem M is solved. Assume this is not the case.

– Then, an ordered sequence of continuous linear programming

problems C_k called *sub-problems* is built and an optimal solution is calculated for each of them.

A sub-problem C_k differs from C by the bound constraints (2.3) which are made more restrictive. Two sub-problems differ at least by one bound constraint over one y_j.

It is convenient to represent this sequence of C_k by a tree composed of *nodes* connected by directed *branches.*

To each node of the tree are attached

– a *continuous sub-problem*, designated by its order number k

– an *optimal solution* of this sub-problem

– *the objective function value* $\overline{F}_k$ of this solution called *functional value* of *node k.*

Note that:

– The continuous problem C is sub-problem 1, attached to node 1 which is the root of the tree.

– A node cannot be defined by an infeasible sub-problem.

– Node k is *integer* if its corresponding optimal solution is an integer solution of sub-problem C_k (thus it also is an integer solution of M).

2.3. *Tree generation – Branching process*

As it will appear later, any non-integer node can generate 0, 1, or 2 new nodes called its *successors.*

Assuming that n nodes have already been created, we shall call *Waiting Set* W_n the set of all the non-integer nodes, also called waiting nodes, which have not yet been used to generate successors. At the beginning of the search, this set will contain only node 1. Let us choose a *waiting node* k among the waiting set W_n. The branching process applied to node k, called *branching node*, consists of:

– Choosing an integer variable y_b called *branching variable*, which has a non-integer value $\bar{y}_b^k$ in the optimal solution of sub-problem C_k. It can be written

$$\bar{y}_b^k = [\bar{y}_b^k] + f_b^k$$

where

$$0 < f_b^k < 1, \quad \text{and} \quad [\bar{y}_b^k] \quad \text{is integer.}$$

– Defining two new sub-problems C_{n+1} and C_{n+2}, which differ from

sub-problem C_k only by the permitted range on the branching variable y_b. The bound constraints

$$\alpha_b^k \leqslant y_b \leqslant \beta_b^k \quad \text{for} \quad C_k,$$

are replaced by

$$\alpha_b^k \leqslant y_b \leqslant [\bar{y}_b^k] \quad \text{for one sub-problem,}$$

$$[\bar{y}_b^k] + 1 \leqslant y_b \leqslant \beta_b^k \quad \text{for the other one.}$$

– Optimizing C_{n+1} and C_{n+2} leads to one of the following events for each sub-problem:

(a) sub-problem *infeasible*, so that no node is created; the corresponding branch is called *infeasible branch*.

(b) sub-problem *feasible* and its optimal solution is *integer*; node ($n + 1$ or $n + 2$) is integer and will never have a successor.

(c) sub-problem *feasible* with a *non-integer* optimal solution; node ($n + 1$ or $n + 2$) is created and taken into the waiting set.

Then, the process continues by choosing another branching node from the waiting set and by applying to it the branching process, and so on. As the integer variables are given finite lower and upper bounds, the generated tree is finite and so is the process. Furthermore, for a given problem, the generated tree is not unique if the choice of the branching variable at a branching node is not unique.

The search is over when the waiting set is empty. At this time each terminal branch is either infeasible or points to an integer node. The best integer solution(s) obtained is (are) optimal. If no integer solution has been obtained, the problem M is unfeasible.

As in most cases this process, although finite, may be time consuming, attempts are made to limit the search. This is the *search limiting process.*

2.4. *Search limiting process*

Its aim is to eliminate forever or provisionally the waiting nodes that are *likely not to lead to the desired results.* We shall call node n *descendant* of node k if it is either its successor or a descendant's successor. From the previous description of the tree generation, a node n, descendant of node k, is defined on a sub-domain attached to node k

(the activity ranges on the integer variables are more restrictive) so that we have $\overline{F}_n \geqslant \overline{F}_k$.

This remark implies the two following properties:

– Assuming integer solutions can be obtained at descendants of node k, $\overline{F}_k$ is a lower bound of the functional value of the best one.

– The value

$$\overline{G}_n = \min \overline{F}_i, \qquad i \in W_n$$

is a lower bound of the best integer solution which can still be expected from the current waiting nodes.

2.5. *Dropping nodes*

A parameter $\overline{F}_\infty$ is provided so that all waiting nodes having a functional value worse than $\overline{F}_\infty$ are abandoned forever. This results in a new definition of the waiting set W_n

$$W_n := \{k \in W_n | \overline{F}_k \leqslant \overline{F}_\infty\}.$$

The value $\overline{F}_\infty$ can be used in several different ways such as:

(1) At the beginning of the search, the user is not interested in integer solutions worse than a specified value.

(2) After one or several integer solutions have been found, the user is not interested in new integer solutions worse than the previous ones (this is the case when only an optimal integer solution is searched for. Parameter $\overline{F}_\infty$ is then set to the functional value of the best one obtained up to now).

(3) The dropping value $\overline{F}_\infty$ can also be used to abandon a subproblem optimization before it is completed if it is proved that the corresponding node would have a functional value worse than $\overline{F}_\infty$.

2.6. *Postponing the processing of nodes*

Another possibility of limiting the tree scanning is to postpone the processing of some waiting nodes. A parameter γ is defined which has the following effect: a waiting node k is provisionally abandoned if $\overline{F}_k > \gamma$. From this definition, a branching node is now selected from a sub-set of the waiting set called the *candidate set* Γ_n.

$$\Gamma_n = \{k \in W_n | \overline{F}_k \leqslant \gamma\}.$$

Parameter γ allows for eliminating provisionally the nodes which are expected to be dropped later or have no interest for the moment but may have one perhaps later. Parameter γ can be used to speed up the search for an optimal integer solution

After the basic method has been introduced, new concepts have now to be presented to define the rules for choosing the branching node and the branching variable, which are the originality of the method.

3. Pseudo-costs and estimations

3.1. *Pseudo-costs*

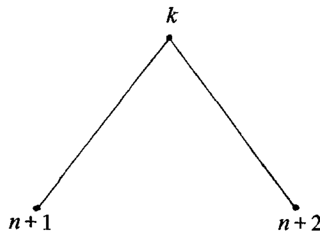
The concept of pseudo-cost is introduced to measure in a quantitative way the "importance" of the different integer variables and to forecast the deterioration of the functional value when forcing an integer variable from a non integer to an integer value. Their justification and their use are based only on experimental results.

Two quantities are attached to each integer variable y_j; they are called lower (PCL_j) and upper (PCU_j) pseudo-costs. At the beginning of the search, pseudo-costs are generally not known; they are automatically computed during tree scanning as follows:

Let us consider the branching process applied at branching node k to branching variable y_b. It can be illustrated in the figure below:

Permitted range: $\alpha_b^k \leqslant y_b \leqslant \beta_b^k$

Current value: $\bar{y}_b^k = [\bar{y}_b^k] + f_b^k$



Permitted range:	$\alpha_b^k \leqslant y_b \leqslant [\bar{y}_b^k]$	$[\bar{y}_b^k] + 1 \leqslant y_b \leqslant \beta_b^k$
Current value:	$[\bar{y}_b^k]$	$[\bar{y}_b^k] + 1$.

Pseudo-costs of integer variable y_b are defined as follows:

$$\mathrm{PCL}_b = \left| \frac{\overline{F}_{n+1} - \overline{F}_k}{f_b^k} \right|, \qquad \mathrm{PCU}_b = \left| \frac{\overline{F}_{n+2} - \overline{F}_k}{1 - f_b^k} \right| .$$

These pseudo-costs appear to be the deterioration of the functional value per unit of change of y_b one corresponding to a decrease and the other to an increase of y_b. Values of pseudo-costs of y_b depend on the node where they are computed.

Nevertheless, experiments performed using real mixed integer problems showed that if pseudo-costs of an integer variable are computed at each tree node where it has a non-integer activity, these pseudo-costs have the same order of magnitude except perhaps at a few nodes. Therefore, in our method, pseudo-costs of an integer variable are assumed to be constant.

3.2. *Estimations*

Using pseudo-costs and values of the integer variables at waiting node k, the functional value of the best integer solution which can be expected at a descendant of node k can be estimated by the following computation:

$$\overline{E}_k = \overline{F}_k + \sum_{j=1}^{q} \min\left(\mathrm{PCL}_j f_j^k, \mathrm{PCU}_j (1 - f_j^k)\right) .$$

This formula assume the pseudo-cost stability and some kind of independence between integer variables. $\overline{E}_k$ is called the estimation of node k.

The estimation of a waiting node is computed when this node is reached. However, if several pseudo-costs were missing (this is true at the beginning of the search), the estimations would be of poor accuracy. Therefore, an optional means is provided to compute missing pseudo-costs at each node if more accurate estimations are desired.

The considerations of pseudo-cost and estimation has led to heuristic rules yielding a method which has appeared to be efficient in the problems we have met.

4. Heuristic rules

In this method the most important rules appear to be the choices of the branching node and of the branching variable. Since it does not seem that theory can decide what the best rules are, experiments have been made with different choices. The program offers a set of rules among which the user can choose. Comparisons between these rules are given in section 7 of this paper.

4.1. *Choice of the branching node*

Let us assume that a branching process is performed at branching node k, so that two new nodes $n + 1$ and $n + 2$ are created. The implementation has been made to have at our disposal all elements related to these two nodes (for example their basis and their corresponding inverse). This is not true for the other waiting nodes. Thus, if one of these two nodes appears to belong to the candidate set, it is convenient to choose it as next branching node. Furthermore, this choice often quickly provides an integer solution, because in such a way the degree of freedom of integer variables decreases monotonously. The functional value of the so obtained integer solution can then be used to drop waiting nodes.

Hence, the first rule governing the choice of the branching node is: If the last branching process has produced only one candidate node, this node is chosen as next branching node.

If two candidates have been obtained, one of them is chosen to be the next branching node according to one of the following criteria:

– choose the node having the best estimation

– choose the node having the best functional value

– choose the node for which the current pseudo-cost of the branching variable at node k is the smaller one.

If the last branching process does not generate any candidate, we have to find one in the candidate set. So that we have the second rule:

– choose the node with the best estimation

– choose the last node created

– choose the last one created until the first integer solution is obtained, then the one having the best estimation.

4.2. *Choice of the branching variable*

Quasi-integer variables

An integer variable y_j is called quasi-integer at node k if its value is

not integer and differs by less than ν from an integer value (ν is generally set to a small positive value, 0.1 for instance).

The choice of the branching variable results from the combination of two distinct rules, one concerning the quasi-integrity of a variable, the other concerning the definition of a priority order.

The branching variable is selected first among integer variables with non quasi-integer values. If there are no such variables, the solution is called quasi-integer, and the branching variable is chosen among quasi-integer variables. This is generally justified, for quasi-integer variables often take integer activities when forcing other integer variables to integer values; so it is not worthwhile to deal with them first.

Priority order

This priority order may be static or dynamic. In the first type, a fixed priority order is given to the integer variables and the branching variable is the first in the list which does not have an integer value. This order can be

– The one provided by the user which should take into account the importance of integer variables in his model, the most important ones being processed first. We will see later that the importance of an integer variable can be measured by the functional value deterioration it entails when forcing it to an integer activity.

– The decreasing order of their absolute cost values in the objective function.

The second type of choice (dynamic priority order) relies on pseudo-costs. The branching variable is the integer variable meeting the following criterion:

$$\max_j \left(\min \left(\mathrm{PCL}_j \cdot f_j^k, \mathrm{PCU}_j \cdot (1 - f_j^k)\right)\right). \tag{4.1}$$

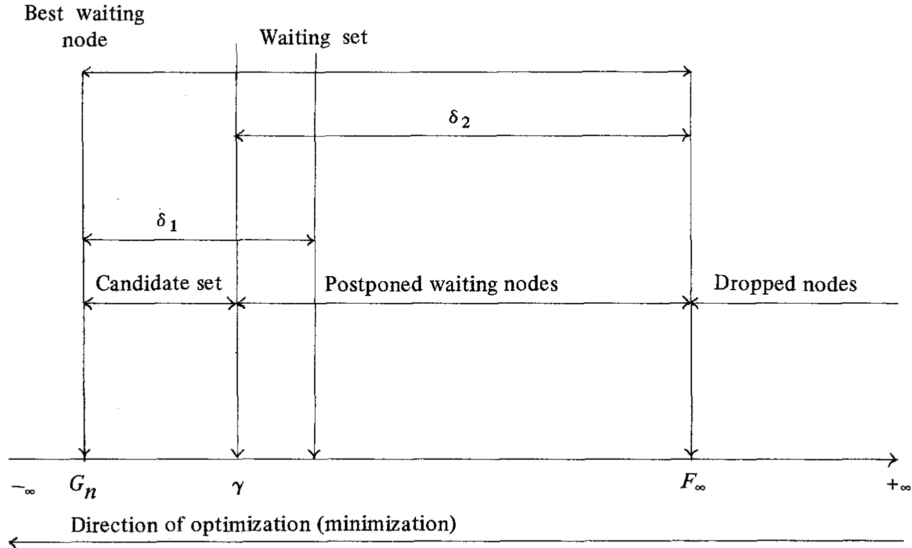
This choice is intended to get the greatest expected deterioration of the functional. The aim could be to violate as fast as possible the upper bound $\bar{F}_\infty$, if it exists.

4.3. *Candidature rules*

We have already defined the candidate set in § 2.6 by a limitation over the functional value of the waiting nodes. This limitation γ can be

defined with regards to the bounds $\overline{G}_n$ and $\overline{F}_\infty$ limiting the waiting set W_n. Let δ_1 and δ_2 be two positive numbers; we use the definition

$$\gamma = \min\,(\overline{G}_n + \delta_1, \overline{F}_\infty - \delta_2)\,.$$



The interest of the 2 parameters δ_1 and δ_2 lies in preventing the search from deeply scanning the tree by not producing nodes which are likely to be dropped when an integer solution will be found. Whereas the first one δ_1 can be mainly used for speeding up the search for an optimal integer solution, the second one δ_2 is important to control the search in order to explore the set of integer solutions according to the user's needs.

Another candidature rule intends to reduce the candidate set with the help of estimations. A parameter $\overline{E}_\infty$ can be set, which has the following effect:

A node k the estimation of which, $\overline{E}_k$, is worse than $\overline{E}_\infty$ is provisionally abandoned and cannot be chosen as branching node. Thus, we can now write a full definition of the candidate set Γ_n.

$$\Gamma_n = \left\{ k \in W_n \;\middle|\; \begin{array}{l} \overline{F}_k \leqslant \gamma = \min\,(\overline{G}_n + \delta_1, \overline{F}_\infty - \delta_2) \\ \overline{E}_k \leqslant \overline{E}_\infty \end{array} \right\}.$$

Parameter $\bar{E}_\infty$ can be used to speed up the search for an optimal integer solution or to give a good presumption that an optimal integer solution has been obtained without continuing the search until optimality is strictly proved.

4.4. *Stopping rules*

Their purpose is to provisionally stop the optimization of a sub-problem before its end when it is expected that the node defined by this sub-problem is likely to be dropped when reached. When the stopping rules are satisfied, the optimization is stopped and a *pre-node* is created to which is attached the sub-problem being optimized. The optimization can be later resumed, if required. Then, the pre-node is put into the waiting set W_n.

One stopping rule has been tested. When node k is quasi-integer (see definition in § 4.2), it is likely that the best expected integer solution will be obtained by rounding the current values of the integer variables at node k to the nearest integers. Thus, if branching node k is quasi-integer, the sub-problem (assume it is C_{n+1}) in which a quasi-integer variable is forced to the nearest integer value is optimized first. Then, the optimization of the other sub-problem will be stopped if it is proved that the functional value of the node to be reached would exceed:

$$\bar{F}_k + 2k(\bar{F}_{n+1} - \bar{F}_k)$$

where k is the number of quasi-integer variables.

5. Optimization process

When applying the branching process to node k, i.e., when creating and solving the two new sub-problems C_{n+1} and C_{n+2}, the following questions arise.

First, knowing an optimal solution of the continuous sub-problem C_k, how to calculate the optimal solutions of C_{n+1} and C_{n+2} respectively; then, how to handle the inverse of the basis in order to minimize the number of required inversions. Now, we shall see how we manage to solve these questions.

5.1. *Optimization of a sub-problem*

Let us consider a branching node k and the branching variable y_{b} the value of which is $\bar{y}_{\mathrm{b}}^{k}$ in the optimal solution of C_k. In that sub-problem, the bounds of y_{b} are: $\alpha_{\mathrm{b}}^{k} \leqslant y_{\mathrm{b}} \leqslant \beta_{\mathrm{b}}^{k}$. Let us assume that we have to optimize sub-problem C_{n+1} which differs from sub-problem C_k only by a restricted activity range for variable y_{b}, which we shall suppose to be:

$$\alpha_{\mathrm{b}}^{k} \leqslant y_{\mathrm{b}} \leqslant [\bar{y}_{\mathrm{b}}^{k}] \ .$$

An obvious way to solve C_{n+1} would be to impose the new upper bound and to solve the sub-problem by means of the simplex dual algorithm. However, we have chosen another method the convenience of which will be explained later on.

We rather parameterize the upper bound of the variable y_{b} and consider the parametric problem derived from C_k by replacing its previous upper bounds β_{b}^{k} by

$$\beta_{\mathrm{b}}^{k} - \theta(\beta_{\mathrm{b}}^{k} - [\bar{y}_{\mathrm{b}}^{k}]) \ .$$

Parameter θ will vary from 0 to 1.

Computational remark

However, since the parameterization with respect to a bound is not a procedure available now in MPSX, we replace the above parametric problem by one in which only the right hand side is a linear function of θ. This is done through an adequate change of variable that will not be described here.

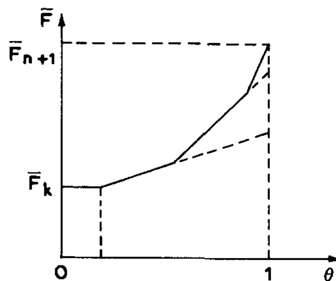


Fig. 5.1.

It is known that, as the parameterization proceeds, the functional value becomes worse and behaves like a convex, piece wise linear, function of parameter θ (see fig. 5.1). At each iteration there generally is a discontinuity in the slope of $\overline{F}(\theta)$. After an iteration has been performed, it is possible to extrapolate the function $\overline{F}$ with the current slope, to get a lower bound of $\overline{F}_{n+1}$, as shown in the figure. This fact is used to either definitely or provisionally abandon the optimization if this lower bound exceeds the dropping value $\overline{F}_{\infty}$ or the stopping value respectively.

5.2. *Inverse of the basis*

In MPSX, the inverse of the basis is expanded in the product from, so that it is composed of an ordered set of elementary vectors. Let us again consider the branching node k, eventually issuing two new nodes, $n + 1$ and $n + 2$. In order to start the optimization of sub-problems C_{n+1} and C_{n+2}, the basis of node k and its corresponding inverse are required.

Before beginning any optimization, the basis of node k is saved and a first pointer indicates the position of the last elementary vector of the current inverse. Then a parameterization is performed to optimize first C_{n+1}, for instance. When the optimization is completed, the current basis and all elementary vectors created during the parameterization are saved, and a second pointer indicates the position of the last created vector. Then it is generally possible to begin the optimization of C_{n+2} by restoring the basis of node k and by retrieving its inverse with the first pointer.

As it was shown in § 4.1, at the end of the second parameterization, we shall try to choose node $n + 2$ or node $n + 1$ as next branching node. If node $n + 2$ is the candidate we can continue with the current basis. On the contrary, if node $n + 1$ is the candidate, its basis can generally be restored and the corresponding inverse be retrieved with the second pointer, and the saved elementary vectors.

6. Some features of the computer program

All particularities of the method described above have been implemented in a module, called MIP, of the IBM Extended Mathematical Programming System (MPSX).

This code can theoretically solve mixed integer problems with up to 16383 rows, not including the bound constraints, and 4095 integer variables. Practically, the size limitation is imposed by the running time.

It is possible to use the program either in a very easy manner or in a more sophisticated one which allows for stopping and resuming the search as well as for defining a search strategy. Furthermore, the parameters of the search limiting process enable the user to explore the set of integer solutions according to his needs.

At last, the user can obtain extensive surveys of the search and perform post optimal studies on the integer solutions found, the integer variables being fixed. Large models (both in number of constraints and number of integer variables) have been successfully solved with this program.

7. Numerical results

7.1. *Global results*

First, two tables (fig. 7.1a and 7.1b) give some results concerning solution of several real-life mixed integer problems together with their characteristics. All problems were solved with the standard strategy,- the rules of which are:

– The integer variables are processed in the decreasing order of their absolute cost values in the objective function (and in the matrix order when absolute cost values are equal).

– The branching node is the one having the best estimation either among the last two candidate nodes just created, if any, or among all nodes belonging to the candidate set.

– The missing pseudo-costs are not computed.

– The tolerance for quasi-integer variable is $\nu = 0.1$.

– Parameters $\overline{F}_\infty$ and γ are set during the search to the functional value of the best integer solution found so far.

For all problems, the mixed stage starts from an optimal continuous solution and ends with an optimal integer solution and the proof of its optimality.

From this problem set, 5 problems have been taken (i.e., 1, 8, 5, 6, 9) in order to compare different strategies and to emphasize the interest of certain choices.

The comparison will be made with both the number of iterations and the number of nodes because these figures do not depend on the com-

Fig. 7.1a
Numerical results

No.	Number of con- straints	Number of variables	Number of non-zero elements	Number of integer variables	Mixed stage CPU time (min)	360 computer	Problem origin
1	38	34	394	14	0.11	75	Investment
2	29	44	329	25	0.13	75	Investment
3	137	172	2941	35	0.93	75	Investment
4	721	1156	20028	39	18.3	75	Production
5	368	397	1592	25	3.11	75	Investment
6	405	340	1920	22	1.05	75	Banking
7	162	183	995	30	0.72	75	Production
8	267	739	2497	14	6.39	75	Plantloc.
9	132	145	1862	28	1.32	75	Production
10	69	590	3377	590	25.7	65	Production

Fig. 7.1b
Numerical results

No.	Number of constraints	Number of variables	Number of non-zero elements	Number of integer variables	Mixed stage CPU time (min)	370 com- puter
11	158	187	868	24	2.02	155
12	28	89	423	30	0.61	155
13	37	74	523	30	1.95	155
14	120	112	768	56	0.49	155
15	157	78	998	78	6.07	155

puter on which the problems were run. Furthermore, the number of iterations has a good correlation with the running time.

7.2. *Estimations*

Comparison of columns 7,1 and 7.2 is intended to show the interest of pseudo-costs and estimations by comparing the standard strategy to a very simple one where no estimation is taken into account. In fact, it differs from the standard strategy only by the choice of the branching node which is:

– Choice of the node having the best functional value among the two candidate nodes just created, if any; otherwise

		7.1		7.2		7.3		7.4	
	Strategy	Standard		No estimation		Order from pseudo-costs		Compute missing P.C. max. degradation	
Problem No.		Number of nodes	Number of iterations	Number of nodes	Number of iterations	Number of nodes	Number of iterations	Number of nodes	Number of iterations
1	Optimal integer solution	44	115	77	244	34	82	76	302
	Proof of optimality	57	176	78	252	44	132	81	324
8	Optimal integer solution	59	1189	76	1600	21	363	19	669
	Proof of optimality	107	2358	112	2465	47	936	34	967
5	Optimal integer solution	108	797	39	204	55	245	35	304
	Proof of optimality	177	1362	176	1341	86	413	95	649
6	Optimal integer solution	38	222	67	416	48	254	42	444
	Proof of optimality	107	698	118	766	57	334	51	482
9	Optimal integer solution	18	105	>250	>826	36	154	59	398
	Proof of optimality	121	452	>250	>826	36	154	>250	>1087

– Choice of the last created candidate node, among all waiting nodes.

The results shown emphasize the importance and the effectiveness of estimations particularly in finding an optimal integer solution. The results for problem 5 seem contradictory, but in this case, with the standard strategy after 31 nodes, an integer solution is found the functional value of which differs from the optimal one by less than 3‰.

7.3. *Order given by pseudo-costs*

Now, the importance of pseudo-costs for obtaining a good priority order for integer variables is pointed out. The two strategies used for the comparisons are on one hand the standard strategy, and on the other hand a strategy identical with the standard strategy except that the priority order of integer variables is still static but is now given by the pseudo-costs. The integer variables can be sorted for instance by decreasing order of the quantity:

$$\max(\mathrm{PCL}_j, \mathrm{PCU}_j)$$

To create such an order, the pseudo-costs have been drawn from previous runs on the same problems.

The comparison of 7.1 and 7.3 clearly show the correlation between the pseudo-cost and the "importance" of the integer variable. The numbers of iterations performed to obtain both an optimal integer solution and the proof of its optimality are significantly reduced.

7.4. *Order and missing pseudo-costs*

In this paragraph, the standard strategy is compared to a strategy which computes the missing pseudo-costs and determines the branching variable in order to obtain the greatest expected deterioration of the functional value as defined in § 4.2. Though the results are unstable, the comparison of 7.1 and 7.4 show the interest of such a strategy to speed up the proof of the optimality, especially for "difficult" problems.

7.5. *Order*

Figure 7.5 shows how large the ratio of running times can be when solving a mixed integer problem with a good and a bad priority order. In both cases the standard strategy is used but for the priority order

Fig. 7.5

Strategy		Good priority order		Bad priority order	
Problem No.		Number of nodes	Number of iterations	Number of nodes	Number of iterations
1	Optimal integer solution	34	82	42	103
	Proof of optimality	44	132	110	365
5	Optimal integer solution	55	245	>250	>1500
	Proof of optimality	86	413	>250	>1500
2	Optimal integer solution	65	107	68	124
	Proof of optimality	71	135	143	329
9	Optimal integer solution	36	154	>250	>1230
	Proof of optimality	36	154	>250	>1230

which is:

$$\max_j \max (PCL_j, PCU_j)$$

for the first case, and the reverse order:

$$\min_j \max (PCL_j, PCU_j)$$

for the second case. The second criterion is so bad that only a few problems have been solved using it. Whereas the order has not always a great effect on the discovery of an optimal integer solution, it is crucial as far as the proof of optimality is concerned. This can be considered as a proof of the significance of pseudo-costs.

As numerical results have shown, the use of pseudo-costs and estimations yields an efficient method. Additional extensive experiment should be carried out on problems with a larger number of integer variables. However, up to now it is possible to infer that:

– The discovery of an optimal integer solution mainly depends on the choice of the branching node.

– The proof of optimality is accelerated by a good choice of the branching variable.

In all cases, the pseudo-costs are reliable to apply these choices.

Acknowledgements

We would like to mention the names of Messrs Battut, Chable, and Leplaideur who participated in a part of the study.

References

[1] E.M.L.Beale and R.E.Small, "Mixed integer programming by a branch and bound technique", Proc. IFIP Congress 65, ed. W.H.Kalenich, Vol. 2 (1966) pp. 450–451.

[2] R.J.Dakin, "A tree search algorithm for mixed integer programming problems", *The Computer Journal* 8 (1965) 250–255.

[3] N.J.Driebeek, "An algorithm for the solution of mixed integer programming problems", *Management Science* 12 (1966) 576–587.

[4] P. Herve, "Resolution des programmes lineaires a variables mixtes par la procedure SEP", *METRA* 6, No. 1–67.

[5] A.H.Land and A.G.Doig, "An automatic method of solving discrete programming problems", *Econometrica* 28 (1960) 497–520.

[6] Carlton E.Lemke and Kurt Spielberg, "Direct search zero-one and mixed integer programming", Report L 3, IBM Data Processing Division, New York Scientific Center (June, 1966).

[7] B.Roy, R.Benayoun and J.Tergny, "De la procédure SEP au programme Ophélie mixte", *SEMA* (1969).

[8] R.Shareshian and K.Spielberg, "On integer and mixed integer programming and related areas in Mathematical programming", IBM N.Y. Scientific Center (1966).

[9] J.A.Tomlin, "Branch and bound method for integer and non-convex programming", NATO International Summer School on integer and non-linear programming (1969).