Un algorithme parallele PR2 du multinapsack

G. Plateau, C. Roucairol, I. Valabregue

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N°811

UN ALGORITHME PARALLELE PR^2 DU MULTIKNAPSACK

Gérard PLATEAU
Catherine ROUCAIROL
Isabelle VALABREGUE

MARS 1988

RR811

ALGORITHM PR^2 FOR THE PARALLEL SIZE REDUCTION OF THE 0.1 MULTIKNAPSACK PROBLEM

UN ALGORITHME PARALLELE PR^2 DU MULTIKNAPSACK

Gérard PLATEAU*, Catherine ROUCAIROL**, Isabelle VALABREGUE***

Abstract

This work exploits the characteristics of parallel machines (vectorization, multiprocessing) in order to perform a lot of tests for the size reduction of the O.1 multiknapsack problem (fixation of variables, estimation of constraints). In the first section of this report, we describe our algorithm for this crucial solving phase, and give first numerical results obtained on a supercomputer CRAY 2. These results are compared with those obtained by previous sequential algorithms, and prove the efficiency of our approach. All details concerning the implementation of the algorithm on the CRAY 2, as well as the information about this supercomputer, the parallel program PR^2 (code in FORTRAN 77, multitasking library cray), and the test problems from literature, are reported in section 2.

Key words : *combinatorial optimization, pseudo boolean optimization, reduction, surrogate relaxation, parallel algorithms, asynchronous multiprocessors machine.*

Résumé

Le but de ce travail est d'exploiter les possibilités offertes par les machines parallèles (vectorisation, multitâches) de façon à augmenter le nombre de tests effectués lors de la procédure tendant à réduire la taille d'un problème de multiknapsack en variables 01 (par fixation de variables, élimination de contraintes). L'algorithme distribué mis au point est décrit dans la première partie. Les premiers résultats obtenus en l'implantant sur un super ordinateur - le Cray 2 - sont comparés à ceux des approches séquentielles, et justifient amplement la démarche proposée. En seconde partie, sont reportés tous les détails concernant son implémentation sur le Cray 2 (description de la machine, de l'algorithme parallèle PR^2 codé en FORTRAN 77 et en utilisant la librairie multitâches CRAY, problèmes tests).

Mots clés : *optimisation combinatoire, optimisation pseudo-booléenne, relaxation "surrogate", algorithme parallèle, machine multiprocesseurs asynchrone.*

Acknowledgement

This research was supported with computer time by CCVR (Centre de Calcul Vectoriel pour la Recherche, Ecole Polytechnique, Palaiseau, France) inside the CAPRAN Research Project (INRIA, Rocquencourt, France).

* Université Paris-Nord, LIPN

** INRIA, Projet Capran

*** Ingénieur IIE, stagiaire INRIA 87

SECTION 1

A DISTRIBUTED ALGORITHM FOR THE SIZE REDUCTION OF THE 0.1 MULTIKNAPSACK PROBLEM

Contents :

INTRODUCTION

Although NP-Hard, the 0-1 knapsack problem is well solved in a sequential way for many classes of instances. The lot of exact algorithms in literature includes implicit enumeration procedures with an experimental linear time complexity (see for example 1,2,13,14,17]). Sizes up to several thousand variables are easily reached for some instances whose data are randomly generated from a uniform distribution.

On the opposite, optimal solutions of the 0-1 multidimensional knapsack problem are classically reached for instances with strongly limited sizes : about ten constraints and one hundred variables [3,4,5,6,15,16,21,22,24].

Up to now, parallel algorithms devoted to the knapsack problem have only been designed for the one dimension case [8,10,11,25]. In addition, all these works are based upon the use of theoretical models of parallel computation (PRAM with a polynomial - even exponential in the input size - number of processors).

The aim of our work is to elaborate an efficient parallel algorithm for solving the 0-1 multiknapsack problem and to implement it on an actual supercomputer whose number of processors is obviously independent of the input sizes.

The solving of the 0-1 multiknapsack problems includes two main phases :

- the reduction of the size ;
  computations of lower bounds (heuristic methods) and upper bounds (Lagrangean and surrogate relaxations) allow to fix variables at there optimal values and to drop redundant constraints
- the implicit enumeration of the reduced problem.

Parallel branch and bound algorithms have been already designed for various combinatorial optimization problems [12,19,20] ; due to the crucial impact of the first phase, we choose to focus our study on this preprocessing reduction algorithm.

The basic idea of this paper is to exploit the speed of supercomputers in order to perform much more works than in a sequential way and thus to improve the size reduction (that is to increase the number of fixed variables and eliminated constraints).

After the description of the reduction tools (section 1), we give in section 2 the main characteristics of our parallel algorithm - denoted by PR^2 (Plateau Roucairol Parallel Reduction). This algorithm is based upon the use of a bounded number of processes which behave concurrently and communicate by exchanging messages. The necessary synchronization and communication operations are specified in section 3.

Algorithm PR^2 has been implemented on the asynchronous multiprocessor machine CRAY 2. The first computational results are summarized by considering classical test problems in section 4.

1. REDUCTION

1.1. Problem statement

The following 0-1 multiconstraint knapsack problem (B) is considered :

$$\begin{array}{ll} \text{maximize} & cx \\ \text{subject to} & Ax \leq b \; ; \; x \in V, \end{array}$$

whose data are such that
$c \in IN_*^n$, $b \in IN_*^m$, A is a mxn dense non-negative integer matrix, and where $V = \{x \in IR^n \mid x_j = 0 \text{ or } 1 \; , j = 1, 2, \ldots , n\}$.

1.2. Notations

$\lfloor \lambda \rfloor$: integer part of a real number λ.
Given a set J :
[J] : convex hull of J ;
| J | : cardinality of J ;
J\U : complement of a given subset U of J.
Given an optimization problem (P) :
v(P) : optimal value of (P) ;
$\bar{v}(P)$ (resp. $\underline{v}(P)$) : upper (resp. lower) bound on v(P) ;
(P | x∈X) : (P) with the added constraint x∈X.
When (P) is a 0-1 problem :
$(\bar{P})$: problem (P) when [V] is substituted for V.

1.3. Reduction tests

The use of a preprocessing reduction algorithm improves the efficiency of exact methods by decreasing the size of the problems (fixation of variables and elimination of constraints) (see [4,5]).

The classical concept of surrogate constraints introduced by Glover [7] allows to generate reduction tools with expected linear time complexities (use for the more sophisticated tests of the solving of knapsack relaxations (see [2,4])).

Before to give the statement of these duality based reduction tests, the simple following ones allow the construction of a so-called *well-stated* problem.

1.3.1. The well-stated problem

This problem is obtained by dropping obviously redundant constraints and fixed variables

(R_1) if there exists an index $j \in \{1,2,...,n\}$ and an index $i \in \{1,2,...,m\}$ such that :

$$a_{ij} > b_j$$

then the variable x_j must be fixed at 0.

(R_2) if there exists an index $i \in \{1,2,...,m\}$ such that :

$$\sum_{j=1}^{n} a_{ij} \leq b_i$$

then the constraint must be eliminated.

(R_3) if there exists an index $j \in \{1,2,...,n\}$ such that :

$$A^j = 0$$

then the variable x_j must be fixed at 1.

1.3.2. Duality based tests

For a well-stated problem (B) with n variables and m constraints, given a multiplier $w \in \mathbb{R}_+^m$ (generated or not by the solving of the Lagrangean (or surrogate) dual of (B)), let us consider :

- the associated surrogate relaxation (0-1 knapsack problem derived from (B)) :

$$(B^o(w)) \qquad \max \; cx \quad \text{s.t.} \; wAx \leq wb \; ; \; x \in V$$

- for each q in {1,2,...,m} , the following 0-1 knapsack problem

$$(B^q(w)) \qquad \max \; A_q x \quad \text{s.t.} \; wAx \leq wb \; ; \; x \in V$$

a) **Fixation of variables**

With the aim of finding a solution with a value greater than a given lower bound $\underline{v}(B)$ on $v(B)$, the fixation of variables at their optimal values is realized as follows :

Theorem 1 :

If there exists an index $j \in \{1,2,...,n\}$ and $\varepsilon \in \{0,1\}$ such that

$v(B^0(w) \mid x_j = \varepsilon) \leq \underline{v}(B)$,

then the variable x_j must be fixed at the value $1- \varepsilon$.

Proof : see [4,16].

b) **Elimination of constraints**

The following result gives a sufficient condition for dropping redundant constraints :

Theorem 2 :

If there exists an index q in $\{1,2,...,m\}$ such that

$v(B^q(w)) \leq b_q$

then the constraint $A_q x \leq b_q$ can be eliminated.

Proof : see [4,16].

1.3.2 Main features of the serial algorithm

The serial reduction system FPR83 realized by Fréville and Plateau includes three main phases :

(i) computation of an upper bound on the optimal value by using Lagrangean and surrogate relaxations.

(ii) computation of a lower bound by using the so-called AGNES heuristic methods.

(iii) size reduction by the conjunction of tests applied to 0-1 knapsack problems derived from (B) as described in theorems 1 and 2.

This serial algorithm FPR83 is extensively described in [4,5] with the actual chain of reduction tests and additional details about the selected options for its implementation.

It has been tested on a CII HB IRIS 80 with twenty concrete problems [3,15,16,21,24] and thirty randomly generated problems [22]. These computational results prove the quality of performances of heuristics AGNES and the decreasing of the running times for exact methods including this preprocessing reduction system FPR83.

For example, for the seven problems due to Petersen [15], table 1 details for each of them their original and reduced sizes, and the running times (in second) for their exact solving by Shih's code [22] respectively without (original size) and with (reduced size) the inclusion of FPR83.

original size	time (second)	reduced size	time	
			reduction	global
10 × 6	.2	0 × 0	1.2	1.2
10 × 10	.5	1 × 3	2.3	2.3
10 × 15	1.9	5 × 8	4.8	4.8
10 × 20	1.8	2 × 8	5.7	5.8
10 × 28	9.2	2 × 9	5.4	5.5
5 × 39	> 25	4 × 28	7.5	25
5 × 50	> 25	4 × 38	7.7	28.8

table 1

2. HIGH-LEVEL DESCRIPTION OF THE ALGORITHM

Instead of constructing the optimal dual multiplier w^* of (B), the aim of our parallel algorithm is to generate two sets W_1 and W_2 of dual multipliers which allow to produce a set of 0-1 knapsack problems :

. *fixation of variables :*

$(B^0(w))$ max cx s.t. $wAx \leq wb$; $x \in V$, $w \in W_1$.

. *elimination of constraints :* $q \in \{1,...,m\}$

$(B^q(w))$ max $A_q x$ s.t. $wAx \leq wb$; $x \in V$, $w \in W_2$.

This idea is motivated by the two following observations.

First, it is not proved that the use of the unique 0-1 knapsack problem $(B^0(w^*))$ leads to the best results as concerns the number of fixed variables and eliminated constraints. In addition, the relation

$$\bar{v}(B^0(w)) \leq \bar{v}(B^0(w')) \quad w,w' \in IR^m_+$$

does not imply necessarily for any variable x_j and $\varepsilon \in \{0,1\}$

$$\bar{v}(B^0(w) \mid x_j = \varepsilon) \leq \bar{v}(B^0(w') \mid x_j = \varepsilon).$$

Second, from a computational time point of view, it is not realistic to consider serially a set of the *tool knapsacks* $(B^0(w))$, $w \in W_1$. However, this kind of approach had been used by Hansen and Plateau [16] in order to test experimentally its robustness.

Obviously, due to parallelism, it is possible to work concurrently with several *tool knapsacks* and to apply a sequence of reduction tests for each of them.

The distributed work is the following :

- *fixation of variables* :

given a multiplier $w \in W_1$, perform the usual sequence of tests on $(B^0(w))$.

- *elimination of constraints* :

given a constraint $q \in \{1,2,...,m\}$ candidate for elimination, use of another sequence of tool knapsacks $(B^q(w))$, $w \in W_2$.

In fact, the 0-1 knapsack problems $(B^0(w)) \mid x_j = \varepsilon)$ and $(B^q(w))$ are not exactly solved, but *with respect to theorems 1 and 2* these different kinds of relaxations are used :

. Lagrangean relaxations

. the associated linear programs.

Namely, for the fixation of variables, the following results are applied :

given a tool knapsack $(B^0(w))$, $w \in W_1$, let us define its Lagrangean relaxation associated with a multiplier $\lambda \in IR^+$:

Theorem 3 :

(i) the function $\lambda \rightarrow v(LR(\lambda))$ is a convex piece-wise linear function.

(ii) the breakpoints of this function are $\lambda_j = c_j / wA^j$, $j = 1,2,...,n$.

Due to theorems 1 and 3, this result is deduced :

Corollary :

For the fixation of any variable of (B), it is sufficient to consider the finite set of Lagrangean relaxations $(LR(\lambda))$ $\lambda \in \Lambda = \{0,\lambda_1,...,\lambda_n,+\infty\}$.

Proofs : see [9].

One more time, instead of constructing the optimal Lagrangean dual multiplier λ^* of $(B^0(w))$, we generate the set Λ of multipliers ; this leads to an obvious vectorizable reduction structure.

Let us recall that for each λ in Λ, by denoting :

$J^+(\lambda) = \{j \in \{1,...,n\} \mid c_j - \lambda wA^j > 0\}$; $J^-(\lambda) = \{j \in \{1,...,n\} \mid c_j - \lambda wA^j < 0\}$

then

$$v(LR(\lambda)) = \lambda wb + \sum_{j \in J^+(\lambda)} (c_j - \lambda wA^j)$$

and for j in $J^+(\lambda)$:

$v(LR(\lambda) \mid x_j=1) = v(LR(\lambda))$; $v(LR(\lambda) \mid x_j=0) = v(LR(\lambda)) - (c_j - \lambda wA^j)$

and for j in $J^-(\lambda)$:

$v(LR(\lambda) \mid x_j=0) = v(LR(\lambda))$; $v(LR(\lambda) \mid x_j=1) = v(LR(\lambda)) + (c_j - \lambda wA^j)$

Thus, for each variable x_j , $j \in J^+(\lambda) \cup J^-(\lambda)$, it is sufficient to exploit the comparison of these two vectors :

$$(|c_j - \lambda wA^j|)_{\lambda \in \Lambda} \qquad \text{and} \qquad (v(LR(\lambda)) - \underline{v}(B))_{\lambda \in \Lambda}.$$

Finally, as concerns the elimination of constraints, in this previous work, we choose to solve the linear programs associated with $(B^q(w))$, $w \in W_2$ by applying the expected linear time complexity algorithm described in [2].

3. ALGORITHM PR2

Our algorithm is based upon the use of a bounded number of processes which are to be dealt with concurrently : they execute different instructions over different data at any time and communicate by exchanging messages through a fixed interconnection network.

Among these N processes, one specialized process, the controller process called S, distributes and schedules the work between the other processes (current process or task-k , k=1,...,N-1).

This allows us to implement this algorithm more easily on a shared memory asynchronous multiprocessors machine.

Each current process performs successively two phases : fixation of variables (phase 1) and constraint elimination (phase 2).

During phase 1, current processes proceed in parallel to the fixation of variables : each of them works with a particular tool knapsack generated by a multiplier w. All the results produced by these processes are then gathered. The cardinality of the set CP of cooperating processes depends on the number of multipliers $|W_1|$ than can be generated :

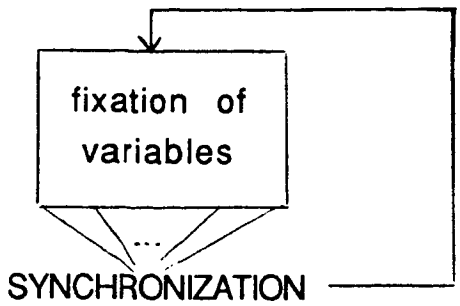
$$|CP| = \min (N-1, |W_1|).$$

This iteration will be performed until all the multipliers have been assigned to processes.

All processes then enter in phase 2, but only if the fixation of variables is no longer efficient : that is to say if at least one of the variables have been fixed at the current iteration of phase 1, |CP| processes will reenter in a new iteration with an initial set W_1 of multipliers.

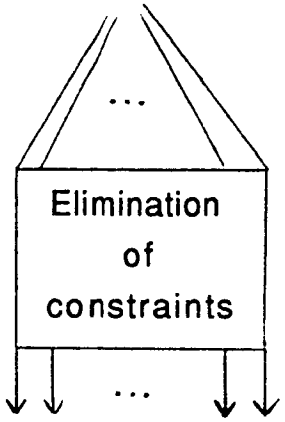
During phase 2, on the contrary, processes are not synchronized : each process works with a candidate constraint q for elimination ($A_q x \leq b_q$) associated with a tool constraint ($wAx \leq wb$, $w \in W_2$) received from the controller process S. Then it sends back its results to process S which returns new work if remains any.

PHASE 1 :



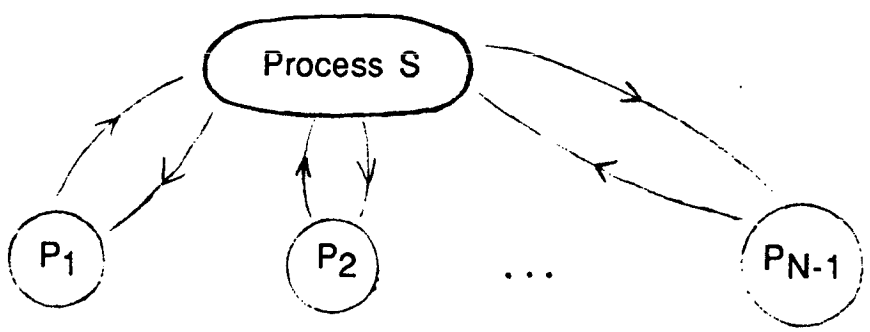
PHASE 2 :

(asynchronous)



The graph of communications between processes during the algorithm is :

Current processes



The structure of messages is (x,y,z) with

- x : type of message,
- y : value of results,
- z : number of process which communicates with S.

Process k

read messages;
upon receiving message ('work1',w,k)
/phase 1/
use the Lagrangean relaxations of
max cx s.t. wAx ≤ wb , x ∈ V ;
update X_0 , X_1 ;
if (all variables are fixed) or (incompatibility detected)
then send message ('stop',(X_0,X_1),k) ;
halt Process k
else send message ('work1',ø,k)
endif
upon receiving message ('nowork',ø,k)
send message ('results',(X_0,X_1),k)
upon receiving message ('stop',ø,k)
halt Process k ;
upon receiving message ('work2',(q,w),k)
/phase 2/
solve the linear program associated with
max A_qx s.t. wAx ≤ wb , x ∈ V ;
if (constraint q is eliminated)
then send message ('new candidate constraint',q,k)
else send message ('new tool constraint',q,k)
endif;
exec Process k ;

Process S

```
/phase 1/
/W1 = set of multipliers W/
reduction ← true ;
generate a well-stated problem (tests R1, R2, R3) ;
while reduction do
        reduction ← false ;
        W ← W1 ;
        CP = set of cooperating processes with |CP| = min(|W1|,N-1),
        for each process k in CP do
            choose w in W ; W ← W\w ;
            send message ('work1',w,k) ;
        endfor ;
    while CP ≠ ø do
        read messages ;
        upon receiving message ('work1',ø,k) :
        if W ≠ ø then  choose w in W ; W ← W\w ;
                       send message ('work1',w,k) ;
                else send message ('nowork',ø,k)
        endif ;
        upon receiving message ('results',(X0,X1),k)
            CP ← CP\k ;
            update X0,X1 ;
            if all variables are fixed or incompatibility detected then
                for each process k in CP do
                send message ('stop',ø,k)
                endfor ;
                halt process S ;
            endif ;
        upon receiving message ('stop',(X0,X1),k)
            CP ← CP\k ;
            update X0,X1 ;
            for each process k in CP do
                send message ('stop',ø,k)
            endfor ;
            halt process S ;
    endwhile ;
```

```
        if at least one variable is fixed
        then    generate a well-stated problem ;
                reduction ← true
        endif ;
endwhile ;
/phase 2/
M = set of constraints ;
/W2 = set of multipliers/
/initialization/
for q=1 to m do W(q) ← W2 endfor ;
for k=1 to N-1 do
        send message ('work2',(q,w),k) with (q,w) ∈ M × W(q) ;
        update W(q) ; status(k) ← 1 /status(k)=1 if Process k is working, 0 otherwise/
endfor ;
                    N-1
while M ≠ ø or      Σ  status(k) ≠ 0 do
                    k=1
        read messages ;
        upon receiving message ('new candidate constraint',q,k)
                                update the results / contraint q is eliminated / ;
                                if q ∈ M then M ← M \ q endif ;
                                if M ≠ ø then
                                        choose q in M with W(q) ≠ ø ;
                                        choose a multiplier w in W(q) ; W(q) ← W(q)\w;
          NEW                           send message ('work2',(q,w),k)
        CANDIDATE               else
                                        send message ('stop',ø,k) ; status(k) ← o ;
                                           N-1
                                        if ( Σ  status(k) = 0) then halt Process S endif
                                           k=1
                                endif
        upon receiving message ('new tool constraint',q,k)
                                if W(q)≠ø then  choose w in W(q) ; W(q) ← W(q)\w ;
                                                send message ('work2',(q,w),k) ;
                                else    update the results /constraint q is not eliminated/ ;
                                        NEW CANDIDATE
                                endif
endwhile ;
```

The algorithm will terminate (Process S halts) when all Processes k , $k \in \{1,\ldots,N-1\}$ halt and when all the constraints $q \in \{1,\ldots,m\}$ have been studied.

4 . EXPERIMENTAL RESULTS

4.1. Implementation

Algorithm PR^2 [18] has been implemented on the asynchronous multiprocessor CRAY2. Its four vector processors communicate by simultaneous reading or exclusive writing on a shared central memory (32 M words, CPU cycle time 4 ns for vector operations).

The adaptation of our distributed algorithm is straightforward : write and read operations on the memory substitute for sended and received messages to (from) the process S.

The code has been written in Fortran 77 by I. Valabrègue [23] and the multitasking library of CRAY has been used for synchronization and communication operations.

The fixation of variables realized by comparing vectors (extensive use of Lagrangean relaxations of 0-1 knapsack problems) has been designed in order to take advantage of automatic vectorization (it concerns the do-statements which do not include any if or goto-statement).

4.2. Test problems

We carry out the test with fifty problems from literature reported in table 2, in order to realize comparisons with the sequential algorithms.

Author	Number of problems	Size from	Size to
Shih [22]	30	5 × 30	5 × 90
Ness, Weingartner [24]	8	2 × 28	2 × 105
Petersen [15]	7	10 × 10	10 × 28
		5 × 39	5 × 52
Senju, Toyoda [21]	2	30 × 60	30 × 90
Hansen, Plateau [16]	2	4 × 28	4 × 35
Fleisher [3]	1	10 × 20	10 × 20

table 2

4.3. Numerical results

First results over 32 problems have been obtained on an emulator of CRAY - called CREM - running on a MULTICS machine BULL DPS 68.

For these computational experiments, we choose to work with tool knapsacks associated with the trivial multipliers : unit vectors of IR^m. Namely, each $B^0(w)$ or $B^q(w)$ are of these types :

max cx s.t. $A_p x \leq b_p$; $x \in V$

max $A_q x$ s.t. $A_p x \leq b_p$; $x \in V$ with $p \in \{1,\ldots,m\}$.

Table 3 contains comparisons of the number of variables fixed during phase 1 by our algorithm PR^2 and a version which only uses the optimal Lagrangean multiplier for each tool knapsack (as in the sequential algorithm of Hansen and Plateau). A strict improvement can be observed for 15 problems (the greatest decrease is 6).

The two next columns include the reduced sizes respectively obtained for phases 1 and 2 by algorithm FRP83 and ours.

Problem	Initial size	Number of remaining variables (phase 1)		reduced size (phases 1 and 2)		time ms CRAY2
		optimal multiplier	PR^2	FPR83	PR^2	
Shih	5 × 70	27	27	3 × 13	3 × 27	105
	-	19	18	1 × 2	3 × 18	117
	-	23	19	3 × 9	5 × 19	115
	-	18	7	1 × 7	1 × 7	116
	5 × 80	22	20	3 × 29	3 × 20	141
	-	20	20	2 × 13	3 × 20	141
	-	13	13	3 × 20	3 × 13	135
	-	20	18	3 × 10	4 × 18	139
	5 × 90	24	22	3 × 30	3 × 22	168
	-	12	12	2 × 7	3 × 12	177
	-	14	14	2 × 8	3 × 14	174
	-	13	13	2 × 9	2 × 13	173
	-	5	0	0 × 0	0 × 0	31
Ness	2 × 28	10	4	1 × 4	1 × 4	31
Weingartner	-	11	11	1 × 5	2 × 11	28
	-	21	21	2 × 19	2 × 21	25
	-	5	5	1 × 5	1 × 5	31
	-	0	0	0 × 0	0 × 0	31
	-	8	6	2 × 7	2 × 6	29
	2 × 105	62	61	2 × 12	2 × 61	106
	-	105	102	2 × 29	2 × 102	70
Petersen	10 × 6	0	0	0 × 0	0 × 0	18
	10 × 10	8	3	1 × 3	1 × 3	24
	10 × 15	8	8	5 × 8	2 × 8	38
	10 × 20	10	10	2 × 8	1 × 10	43
	10 × 28	13	10	2 × 9	1 × 10	55
	5 × 39	38	38	4 × 28	5 × 38	45
	5 × 50	47	46	4 × 38	5 × 46	68
Senju	30 × 60	53	53	30 × 40	30 × 53	574
Toyoda	30 × 90	51	51	30 × 37	30 × 51	556
Hansen	4 × 28	28	28	4 × 27	4 × 28	23
Plateau	4 × 35	35	34	4 × 34	4 × 34	33

Table 3

Given a tool knapsack, we have to point out that the sequential algorithm FPR83 makes use of more elaborated tests for the variable fixation (solving of linear programs with sometimes additional alternative fixation of optimal basic variable - first level of a branch and bound tree) as well as for constraint elimination (solving of linear programs at first or two levels of a b.b. tree).

Furthermore, we recall that Fréville and Plateau use near optimal dual multipliers of (B) in order to construct "good" tool knapsack problems.

Nevertheless, our method is at least as good for 15 problems (with a strict improvement for seven of them).

The last column details CPU times in milliseconds for our algorithm (with 4 processes) implemented on the CRAY2 working with only one processor. This allows to show that even the simulation of parallelism leads to very small computational times : from .180 second (for Petersen's problem 1, 10×6 reduced to 0×0) to .574 second (for Senju and Toyoda's problem 1, 30×60 reduced to 30×53).

Besides these times include the data reading (about 30 per cent of global time !) and may differ from one execution to another due to non determinism.

Additional experiments have been performed on CRAY2 in order to measure the impact of using more than one processor. In table 4, for the two problems of Senju and Toyoda, we report the values of the following parameters :

T_j : the time with j processors (computed from CRAY2 informations, i.e. CPU times t_i with exactly i working processors, $T_j = \sum_{i=1}^{j} t_i$).

S_j : the experimental speed-up with j processors measured by the ratio : T_1 / T_j.

Size	j	Tj ms	Sj
30 × 60	1	462	1
	2	307	1.50
	3	277	1.67
	4	332	1.39
30 × 90	1	441	1
	2	332	1.33
	3	356	1.24
	4	558	.79

Table 4 - Senju and Toyoda's problems.

As the initialization, management and interaction of tasks are accomplished by code that is not in the original program, this additional execution time limits the granularity of parallelism that can be profitably exploited. The costs of multitasking and the size of tasks that can benefit from parallel executions must therefore be appreciated.

These examples illustrate the fact that for small sizes of instances the overhead penalty - amount of time spent for activating and creating parallel tasks added to access to shared central memory ... - has not to be neglected. It can also outweight the amount of actual reduction work on each processor. In this case, it proves that balanced tasks must have greater sizes.

It is clear that the use of four processors will be much more attractive for instances with a number of variables greater than those of literature.

CONCLUSION

The results of our first experiments indicate that :

- more elaborated reduction tests must be implemented in order to increase the numbers of fixed variables and eliminated constraints. In addition, this greatest amount of work by process will lead to a better exploitation of the supercomputer,

- a lot of good dual multipliers must be generated in order to construct good surrogate constraints for the tool knapsack problems.

Finally, this fast improved implementation of our parallel reduction algorithm PR^2 will be very attractive and must take place in any parallel procedure for the exact solving of great size instances of the multiknapsack problem. This will be the future direction of our work.

SECTION 2

UN ALGORITHME PARALLELE PR^2 POUR LA REDUCTION DU MULTIKNAPSACK : IMPLEMENTATION SUR UNE MACHINE QUADRI-PROCESSEURS ASYNCHRONE A MEMOIRE PARTAGEE - LE CRAY 2 -

Sommaire :

1. PRESENTATION DU CRAY 2

Le CRAY 2 est composé de 4 processeurs "d'arrière plan" capables d'effectuer des opérations vectorielles et scalaires, d'une mémoire commune de trés grande taille (soit 268 435 456 mots) et d'un processeur "d'avant plan" invisible à l'utilisateur et dont la tâche est d'assurer la supervision du système, les entrées-sorties et la synchronisation des 4 processeurs d'arrière plan en cas de travaux utilisant le multitâche

Le cycle de base du CRAY 2 est de 4,1 ns, à comparer aux 12,5 ns du CRAY 1, et aux 9,5 ns du CRAY_XMP. Les opérations vectorielles produisent un résultat tous les cycles. Cependant, pour les autres calculs il faut compter un cycle effectif égal à deux cycles de base, soit 8 ns.

L'environnement logiciel du CRAY 2 est donné par 2 compilateurs : le compilateur CFT2, dérivé du compilateur CFT09 pour CRAY 1, qui fonctionne actuellement sans avoir encore atteint une stabilité tout à fait satisfaisante; le compilateur CFT77, nouveau compilateur CRAY, beaucoup plus performant que le premier, mais pour l'instant moins rapide en raison de sa très récente mise en service.

Le système d'exploitation basé sur UNIX System V d'ATT, conçu initialement pour gérer des mini-ordinateurs, pose un certain nombre de problèmes d'adaptation et d'exploitation qui ne sont pas tous résolus .

1.1 - la conception d'algorithme pour machine multiprocesseur vectoriel asynchrone de type CRAY 2

Les difficultés inhérentes à la conception d'algorithme sur une telle machine parallèle résident dans la synchronisation des tâches et la gestion des conflits d'accès aux données partagées.

1.1.1 - le multitâche

Le multitâche permet l'exécution en parallèle, sur différents processeurs, de plusieurs parties d'un même programme. Ces parties se composent d'une suite d'instructions qui elles, sont exécutées séquentiellement.

Le programmeur devra donc concevoir un algorithme en le découpant en tâches, et en tenant compte du fait que le multitâche ne peut garantir que les tâches soient exécutées dans un ordre particulier, et qu'une tâche donnée se termine la première. Le multitâche est non déterministe par rapport au temps. Il est alors nécessaire d'utiliser des mécanismes de synchronisation et de communication entre les tâches.

1.1.2 - Synchronisation et communication

La synchronisation permet de définir des points de rendez-vous entre plusieurs tâches, durant leur exécution.

Ce mécanisme est généralement employé lorsqu'une tâche veut s'assurer que des variables ont bien été calculées par une autre tâche avant de les utiliser dans la suite du traitement.

Les points de synchronisation sont spécifiés par l'appel de primitives système. Il est possible de synchroniser globalement les tâches entre elles, grâce aux primitives telles que "lancer une tâche" (TSKSTART) ou "attendre la fin d'une tâche" (TSKWAIT), ou bien de synchroniser seulement une séquence d'instructions avec des évènements.

La communication entre tâches n'est possible que par l'intermédiaire d'une mémoire commune. Les variables partagées entre les différentes tâches nécessitent des mécanismes de protection. Les machines de type CRAY2 permettent l'emploi de la notion classique de sémaphore pour protéger les sections critiques.

Les primitives de synchronisation

L'algorithme est destiné à être implémenté sur CRAY 2 et éventuellement SEQUENT. Nous disposons, pour exprimer l'exclusion mutuelle des primitives:

LOCKASGN identifie un nom de variable de type entier comme un verrou

LOCKON verrouille une section critique et rend la main à la tâche qui a fait la demande

LOCKOFF déverrouille une section critique et rend la main à la tâche qui a lancé l'appel

et pour simuler un rendez-vous des primitives de gestion des évènements:

EVASGN identifie une variable de type entier comme un évènement

EVWAIT attend jusqu'a ce qu'un évènement spécifique soit envoyé. Si l'évènement est déjà envoyé, la tâche n'attend pas

EVPOST envoie un évènement et rend le contrôle à la tâche qui a lancé l'appel; les tâches en attente peuvent reprendre leur exécution.

EVCLEAR efface l'envoi d'un évènement et rend la main à la tâche qui a effectué l'appel; les tâches qui, par la suite exécutent EVWAIT seront effectivement mises en attente.

La machine SEQUENT permet l'utilisation de primitives de gestion de rendez-vous telles que:

S_INIT_BARRIER identifie une variable qui va être utilisée comme rendez-vous pour exactement n processus

S_WAIT_BARRIER met le processus qui a fait l'appel en attente active jusqu'à ce que n tâches ait lancé cet appel; à ce moment les n tâches sont libérées de l'attente active.

1.1.3 - Etat des tâches

L'attribution d'un processeur physique à une tâche est totalement transparente pour le programmeur. Avant d'obtenir un processeur physique, une tâche doit obtenir un processeur logique par l'ordonnanceur de tâches (scheduler). L'ordonnanceur de tâches est aussi chargé de gérer les files d'attente dues aux "verrous" bloquants, aux évènements bloquants, et celle des processus en attente de processeur logique. C'est ensuite le système (C.O.S. : CRAY Operating System) qui gère le passage d'une tâche d'un processeur logique à un processeur physique.

1.1.4 - Gains attendus

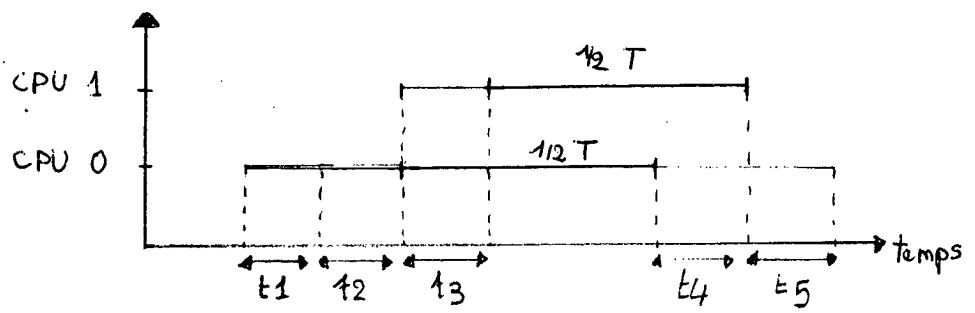
Théoriquement, le temps d'exécution d'un programme devrait dépendre linéairement du nombre de processeurs de la machine. Mais en pratique le parallélisme introduit de nouveaux facteurs (inexistants en mode séquentiel), tels que la synchronisation, et la communication, indispensables, mais qui contribuent à la dégradation des performances espérées.

Le multitâche devient intéressant si le temps gagné par l'exécution de tâches en parallèle est supérieur à celui engendré par la gestion de cette méthode.

Le facteur d'amélioration est calculé de la façon suivante:

$$A = \frac{\text{temps d'exécution avec } \mathbf{1} \text{ processeur}}{\text{temps d'exécution avec } \mathbf{n} \text{ processeurs}}$$

Prenons l'exemple d'une machine biprocesseur sur laquelle les deux tâches exécutent chacune la moitié d'un traitement. L'une d'elles, appelée tâche initiale (similaire à un programme principal), est en plus chargée de lancer l'exécution de l'autre.



t1 : Préparation des paramètres de lancement de la deuxième tâche.
t2 : Appel de la primitive de lancement.
t3 : Prise en compte par CPU 1 de l'ordre de lancement.
t4 : Attente de la fin d'exécution de la deuxième tâche.
t5 : Prise en compte de la fin d'exécution.

Le temps d'exécution total du traitement est donc égal à t1 + t2 + 1/2T + t4 + t5, alors qu'en séquentiel il aurait été de T.

Le facteur d'amélioration vaut :

$$T/(t1 + t2 + 1/2T + t4 + t5)$$

Le multitâche est intéressant dans le cas où on a :
$t1 + t2 + t3 + t4 + t4 + t5 < 1/2T$.

Une bonne amélioration peut être obtenue lorsque les tâches sont de taille voisine. En effet, de cette manière le temps d'attente de fin d'exécution de tâche est minimisé.

D'autre part, la taille d'une tâche doit être suffisante par

rapport au surplus de code généré par le multitâche (appels de primitives). Une formule (voir [LA]) permet de calculer la taille minimale d'une tâche, pour obtenir une amélioration donnée.

Soient
X : taille minimale de la tâche, en tops d'horloge
NCPU : nombre de processeurs
A : amélioration désirée (A<NCPU)
S : temps utilisé pour gérer le multitâche, en tops d'horloge

$$X = \frac{A * NCPU * S}{NCPU - A}$$

On remarque que l'amélioration maximale (A=NCPU) n'est pas réalisable.

1.2 - Vectorisation

1.2.1 - principe

Pour traiter une opération sur un vecteur, les machines classiques sont obligées de passer par l'intermédiaire d'une boucle, dans laquelle l'opération s'effectue séquentiellement sur chacune des n composantes du vecteur. Par contre, les machines vectorielles dispose d'un jeu d'instructions élémentaires (addition, multiplication ...)qui ont été spécialement définies pour les vecteurs, et qui permet de traiter simultanément les n opérations.

Le langage Fortran utilisé pour programmer les machines de type CRAY ne possède pas d'instructions vectorielles. La vectorisation est réalisée de façon automatique par le compilateur Fortran, qui sait reconnaître certaines instructions vectorisables (boucles "DO" particulières).

Le champ d'application de la vectorisation est limité par la règle interdisant toute instruction DO, IF, GOTO, et CALL dans un corps de boucle. Seules les boucles les plus internes sont prises en compte. Les opérations vectorielles sur plusieurs niveaux de boucle échappent totalement au compilateur.

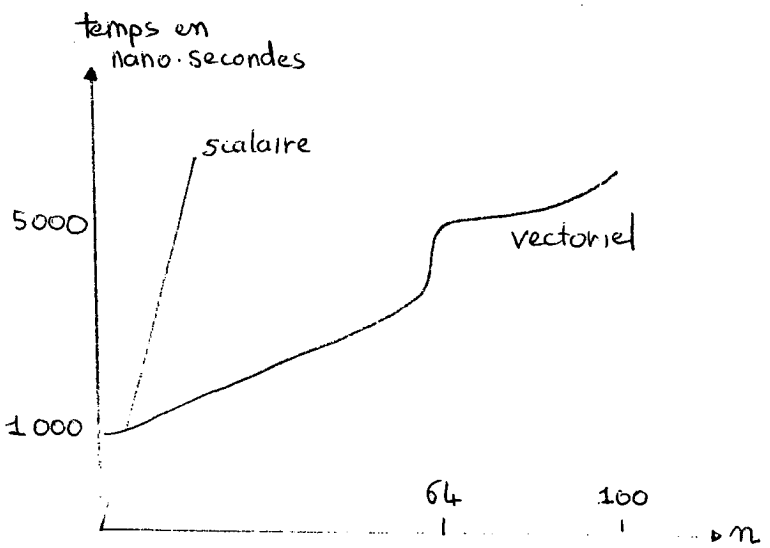
Par conséquent, le programmeur doit apporter beaucoup de soin à la traduction des opérations vectorielles de façon à écrire un code le plus vectorisable possible par le compilateur.

1.2.2 - gains attendus

Jusqu'à l'avènement des machines vectorielles, le temps d'exécution d'une opération sur un vecteur était proportionnel à la longueur n de ce vecteur. Etant donné S le temps d'exécution d'une opération sur deux scalaires, le temps d'exécution de l'opération sur deux vecteurs à n composantes valait nS.

Actuellement, une machine vectorielle comme le CRAY 2, se caractérise par un temps d'exécution de D + nV. V est défini comme S, mais très nettement inférieur grâce à l'emploi du pipeline (S/V ≈10 sur le CRAY 1), et D représente le temps de démarrage.)

L'entier n = D/(S-V), à partir duquel le gain dû à la vectorisation est appréciable, dépend de la machine. Le schéma suivant permet de comparer les temps d'exécution en mode scalaire et en mode vectoriel. Les mesures ont été faites pour l'addition sur le CRAY 1; la vitesse du CRAY 2 est sensiblement identique en scalaire, est peut être double en vectoriel.



Le décrochement à n=64 est dû à la taille des registres vectoriels. Les vecteurs sont stockés par tranches de 64 composantes.

Multitâche et vectorisation sont deux notions complémentaires, dont la combinaison laisse prévoir d'importants gains de temps.

2. ALGORITHME PARALLELE DE REDUCTION DU MULTIKNAPSACK

2.1 - Présentation

2.1.1 - Variables employées

Définition des variables :

Variables globales:

fin_phase_1 la phase 1 est terminée
fin_tâche_k la tâche k est terminée
contradiction_détectée on ne peut pas trouver une solution meilleure que celle associée au minorant donné en entrée
pb_résolu la solution du problème est connue
phase2_efficace au moins une contrainte a été éliminée au cours de la phase 2

NTA : nombre de tâches actives
TC : tableau indiquant l'état des contraintes
CPT : nombre de tâches arrivée au rendez-vous
NBFINI : nombre de tâches terminées
I_EVT : numéro de l'évènement bloquant
FINSEM, CPTSEM, TCSEM sémaphores d'exclusion mutuelle
EVENT(1..2)tableau d'évènements envoyés alternativement lorsque toutes les tâches ont terminé une itération de la phase 1

Variables locales:

no_evt : numéro de l'évènement qui est attendu par une tâche

2.1.2 - Algorithme

Tâche initiale

DEBUT

- Lecture des données

- Initialisation des variables partagées

- Appliquer R1, R2, R3

- Mettre à jour NTA si le nombre de contraintes non éliminées est inferieur au nombre de processus.

- Lancer les tâches "traitement parallèle"

- Exécuter "traitement parallèle"

- Attendre fin d'exécution des tâches "traitement parallèle"

FIN

Tâche "traitement parallèle"

Paramètre transmis : k, le numéro de la tâche

DEBUT

-initialiser les variables locales

Tantque non fin_phase_1 et non fin_tâche_k faire

Tantque non (contradiction_détectée ou toutes les variables sont fixées ou toutes les contraintes sont traitées) faire

LOCKON(TCSEM)
-prendre une contrainte i à traiter
LOCKOFF(TCSEM)
-procédure de fixation de variables:
Calculer $\lfloor v(RB^0{}_i(w_l) \mid x_j = \varepsilon) \rfloor$
$(w_l = c_l/a_l)$, $l \in \{1,2,\ldots,n\}$ et $j \in \{1,2,\ldots,n\}$,
Détecter une contradiction
Mettre à jour les valeurs fixées par la tâche k, du vecteur X solution du programme
Ftque

LOCKON(CPTSEM)
CPT=CPT + 1
no_evt=I_EVT

si CPT≠NTA alors
LOCKOFF(CPTSEM)
EVWAIT(no_evt)

sinon
LOCKOFF(CPTSEM)

-lire les résultats obtenus par chaque tâche
-détecter une contradiction
-détecter si toutes les variables ont été fixées
-détecter si au moins une variable a été fixée
-mettre à jour les composantes fixées à cette itération du vecteur X solution du programme
-appliquer R1, R2, R3 sur le problème réduit
-modifier le vecteur X en conséquence
-mettre à jour le tableau TC, indiquant l'état des contraintes pour l'itération suivante (à traiter, à ne pas traiter, éliminée).
-détecter si le nombre de contraintes non éliminées est inferieur au nombre de tâches actives (NCA<NTA), décider la fin des tâches en surplus.

CPT=0
I_EVT=1-I_EVT
EVCLEAR(EVENT(I_EVT)
EVPOST(EVENT(1-I_EVT))
fsi
Ftque

Si non_pb_résolu et non fin_tâche_k alors

tantque non (toutes les contraintes sont traitées) faire
LOCKON(TCSEM)
-prendre une contrainte i, à traiter: TC(i)=1->TC(i)=2
LOCKOFF(TCSEM)
tantque il reste des contraintes j non éliminées et non utilisées pour éliminer la contrainte i faire
LOCKON(TCSEM)
-choisir j telle que TC(j)≠0
LOCKOFF(TCSEM)
-résolution en continu d'un knapsack:

calculer $v(\overline{B^1}_j)$

si $\lfloor v(\overline{B^1}_j) \rfloor \leq b_i$ alors
LOCKON(TCSEM)
si TC(j)≠0 alors TC(i)=0 fsi
LOCKOFF(TCSEM)
fsi
ftque
ftque

```
    LOCKON(FINSEM)
    NBFINI:=NBFINI + 1
    si NBFINI=NTA et non pb_résolu et phase2_efficace alors
    -appliquer R3 au problème réduit
    fsi
    LOCKOFF(FINSEM)

fsi

FIN
```

2.1.3 - Synchronisation des tâches

En fin de chaque itération de la phase 1, les tâches travaillant en parallèles sont synchronisées, afin que les données du problème réduit soient mises à jour pour l'itération suivante.

Pour éviter une attente active, la synchronisation se fait à l'aide de deux évènements fonctionnant en bascule (l'un des évènements est toujours passant alors que l'autre est bloquant).

Le dernier processus actif en fin d'itération inverse l'état des deux évènements, libérant ainsi les processus bloqués, tout en préparant le bloquage sur l'autre évènement pour l'itération suivante.

Cette technique suppose l'emploi:

-du compteur de tâches CPT reflétant à tout moment le nombre de tâches bloquées
-du sémaphore CPTSEM pour protéger l'accès à cette variable
-de deux évènements EVENT[0] et EVENT[1]
-de la variable globale I_EVT contenant l'indice de l'évènement bloquant à un instant donné

Initialement, l'état des variables est le suivant:

CPT=0
CPTSEM passant
EVT[0] effacé
EVT[1] envoyé

2.1.4 - Procédure de fixation des variables

Introduction :

Il s'agit , ici, de présenter le corps de la boucle constituant le travail d'une tâche pendant la phase 1. Comme on peut le voir, cette étape très calculatoire, est destinée à bénéficier d'une optimisation par la vectorisation.

Rappelons, les données du knapsack (B_p) traité par la tâche k

$$(B_p) \quad \begin{array}{l} \max cx \\ \text{s.c. } A_p x \leq b_p \\ x \in V \end{array} \qquad RB_p(w_i) \quad \begin{array}{l} \max cx + w_i (b_p - A_p x) \\ x \in V \end{array}$$

Soit $\underline{v}(B)$, la valeur d'un minorant de v(B),valeur optimale
Soientt X_0, X_1 ensemble des indices des variables fixées à 0 et à 1 au cours de itérations précédentes.
Soientt X^k_0, X^k_1, ensemble des indices des variables fixées à 0 et à 1 par la täche k pendant l'itération en cours.
Soit $X_2 = \{1..n\} - X_0 - X_1 - X^k_0 - X^k_1$

Les tests d'élimination de variables sont appliqués au problème

$$\max \sum_{j \in X_1 \cup X^k_1,\ a_{pk}=0} c_j + \sum_{j \in X_2,\ a_{pk}\neq 0} c_j x_j$$

$$\text{s.c. } \sum_{j \in X_2,\ a_{pk}\neq 0} a_{pj} x_j \leq b_p$$

Nous donnons:

Procédure de fixation des variables:

CALCUL du vecteur $W=(w_i)$, $w_i = c_i/a_i$;
CALCUL de la matrice $CR=(cr(i,j))$;
$cr(i,j) = c_{pj} - w_i * a_{pj}$
CALCUL de la matrice $M=(m(i,j))$;

$$m(i,j) = \lfloor v(RB^0_p(w_i)) \mid x_j=\varepsilon \rfloor = \lfloor w_i b_p + \sum_{l \in X_1 \cup X^k_1,\ a_{pk}=0} c_{pl} + \sum_{l \in X_2,\ a_{pk}\neq 0} cr(i,l) - |cr(i,j)| \rfloor$$

Tantque ($j \in X_2$, $a_{pk}\neq 0$) et ($\underline{v}(B)$ *non optimale*) *faire*
  CHERCHER un indice i_1 *tel que* $m(i_1,j) \leq \underline{v}(B)$ *et* $cr(i_1,j)\neq 0$
  Si i_1 *trouvé alors*
    Tantque ($i \in X_2$ *et* $a_{pi}\neq 0$ *et non* i_2 *trouvé*) *faire*
      CHERCHER un indice i_2, *tel que* $m(i_2,j) \leq \underline{v}(B)$ *et* $cr(i_1,j) * cr(i_2,j) < 0$
      si i_2 *trouvé alors on ne peut améliorer* $\underline{v}(B)$
      sinon si $cr(i_1,j) > 0$ *alors* $X^k_1 \equiv X^k_1 \cup \{i_1\}$
        sinon $X^k_0 \equiv X^k_0 \cup \{i_1\}$ *fsi*
      fsi
    Ftque
  fsi
Ftque

2.1.5 - Résolution en continu d'un knapsack ([2], Fayard, Plateau)

Introduction : Ce traitement, constitue l'essentiel des calculs effectués par une tâche pendant le déroulement de la phase 2. On trouvera en annexe A1, la fonction SOLVE qui le réalise .

Jusqu'en 1977, $\overline{(B)}$ a toujours été résolu par l'algorithme classique appelé C.K.R. (i.e. Classical Knapsack Relaxation):

Algorithme C.K.R.

Trier les données pour obtenir (après renumérotation des variables)

$c_{j1}/a_{j1} \geq c_{j2}/a_{j2} \geq \ldots \geq c_{jn}/a_{jn}$;

$k^* := min \{ k \mid \sum_{p=1}^{k} a_{jp} > b \}$;

$U := \{ j_1, \ldots, j_{k^*-1} \} \cap \{1, \ldots, n\}$

si $\sum_{j \in U} a_j = b$ alors $L = I - U$

sinon $i := j_{k^*}$;

$L := I - (U \cup \{i\})$;

$\overline{x_i} := (b - \sum_{j \in U} a_j) / a_i$

fsi

$\forall j \in U \quad \overline{x_j} := 1$;

$\forall j \in L \quad \overline{x_j} := 0$;

$v(\overline{B}) := c\ \overline{x}$.

Compte-tenu du temps de calcul prohibitif de la première étape de cet algorithme, le nouvel algorithme envisagé par Fayard et Plateau pour résoudre $\overline{(B)}$ - appelé N.K.R. (i.e. New Knapsack Relaxation) - est une adaptation de la méthode déterminant les p plus grands éléments d'une liste de n nombres (p donné < n), qui est elle même dérivée de l'algorithme de tri Quicksort.

L'exposé de l' algorithme N.K.R. nécessite la donnée de

$k \in J, \quad lN \supset Q \supset J$

$Z = \bigcup_{j \in Q} \{ z_j \}$ où $z_j \in lR, \ \forall j \in Q$

la (Z,k) partition de J est définie par la réunion des trois sous-ensembles de J

$$U(J, Z, K) = \{ j \in J \mid z_j > z_k \}$$

$$E(J, Z, K) = \{ j \in J \mid z_j = z_k \}$$

$$L(J, Z, K) = (J - U(J, Z, K)) \cup E(J, Z, K)$$

Algorithme N.K.R. (Fayard, Plateau, [2])

$J \equiv K$; $U \equiv \emptyset$; $R \equiv \cup_{j \in J} \{ c_j/a_j \}$

trouvé=faux;
Tantque (non trouvé) faire
  Tantque $\sum_{j \in U} a_j > b$ *faire*
    Sélectionner $k \in J$, et déterminer la (R,k) partition de J
    si $\sum_{j \in U} a_j > b$
    alors $J \equiv U(J,R,K)$
    fsi
  Ftque
  $U \equiv U \cup U(J,R,K)$;
  si $\sum_{j \in U} a_j = b$
  alors $L \equiv I-U$;
    trouvé=vrai
  sinon
    si $b < \sum_{j \in U} a_j + a_k$
    alors $i \leftarrow k$;
      $L \equiv I-(U \cup \{i\})$;
      $x_i \leftarrow (b - \sum_{j \in U} a_j)/a_i$;
      trouvé=vrai
    sinon
      si $\sum_{j \in U} a_j \geq b$
      alors *construire* $G \mid E(J,R,K) \supset G$ et $j \in U \cup E(J,R,k)$
        $\sum_{j \in U} a_j \leq b < \sum_{j \in U \cup G} a_j + a_p$, $p \in I-(U \cup G)$;
        $U \equiv U \cup G$;
        si $\sum_{j \in U} a_j = b$
        alors $L \equiv I-U$
        sinon $i \leftarrow p$;
          $L \equiv I-(U \cup \{i\})$;
          $x_i \leftarrow (b - \sum_{j \in U} a_j)/a_i$
        fsi
      sinon $U \equiv U \cup E(J,R,k)$;
        $J \equiv L(J,R,k)$
      fsi
    fsi
Ftque

$\forall j \in U \quad \overline{x_j} := 1 \quad ; \forall j \in L \quad \overline{x_j} := 0 \quad ; \quad v(\overline{B}) = c\,\overline{x}$.

L'algorithme N.K.R. est de complexité $o(|K|)$, ($|K|$=n=nombre de variables), en moyenne.

2.1.6 - Discussion

- Synchronisation des tâches en phase d'élimination des variables

Il apparaît un point de synchronisation au sein de la boucle d'élimination des variables. C'est un rendez-vous qui sert faire une synthèse des résultats obtenus individuellement par chaque processus, et à mettre à jour les données du problème réduit pour la prochaine itération.

Nous avons abandonné l'idée d'une phase 1 asynchrone, pour les raisons suivantes:

1- il nous a semblé que le choix d'un **traitement asynchrone** était **mal adapté au problème** du fait que nous nous attendons à ce que les **traitements effectués en parallèle soient rapides et de longueur équivalente**. De ce fait, d'une part, la solution asynchrone pourrait engendrer un conflit d'accès aux variables partagées, par conséquent une attente très importante dans les sections critiques, par rapport au temps de calcul.
D'autre part, **la solution synchrone qui laisse les tâches exécuter une itération sans être bloquées, ne devrait pas être pénalisante du point de vue de l'attente au point de synchronisation** puisque les tâches arrivent en fin d'itération pratiquement simultanément.

2- Un autre facteur nous a poussé à abandonner la gestion asynchrone du traitement : les tests simples R1, R2, R3 doivent être effectués régulièrement, ils se prêtent mal à une répartition sur les tâches actives. La solution idéale est de les faire exécuter par une seule tâche à chaque fin d'itération. Encore faut-il pouvoir obtenir un état cohérent du système des contraintes. La *synchronisation permet de **fixer** pour un instant un **état du problème réduit**.*

Pour profiter au maximum de la réduction du temps de calcul due au parallélisme, il faut *minimiser le nombre de points de synchronisation*, le nombre de sections critiques et le nombre d'instructions exécutées en section critique.

Dans cette optique, nous avons placé **un seul rendez-vous à l'intérieur de la boucle destinée à éliminer des variables**.
Nous avons envisagé de synchroniser les tâches au fur et à mesure qu'elles traitent une contrainte différente, afin de réduire le temps de calcul des tâches, et éviter des calculs redondants . Nous avons jugé que le gain de temps serait trop faible par rapport au temps perdu en synchronisation.

- Gestion asynchrone de l'élimination des contraintes

Contrairement à la phase 1, la phase 2 est asynchrone. En effet, en phase 2 les besoins de communication entre les tâches sont très différents: au cours d'une itération de la boucle d'élimination des contraintes, les informations requises par une tâche sont les suivantes :
- l'état des contraintes : éliminée, traitée, à traiter pour sélectionner une contrainte i à éliminer
-l'état des contraintes : éliminée, active pour sélectionner une contrainte j - outil pour éliminer i-.

Etant donné que la résolution en continu de nombreux knapsacks est coûteuse, nous avons choisi de minimiser le nombre de knapsack (B^i_j) traités. Pour cela, les tâches ont besoin de l' information la plus récente possible. C'est pour cette raison que les tâches effectuent au minimum deux lectures de l'état des contraintes par itération. Une autre stratégie pourrait consister à minimiser le nombre de lectures en mémoire partagée, en envisageant une seule lecture au niveau de la sélection de la contrainte i.

Pour concevoir un système où les tâches effectuent de nombreuses lectures en section critique et peu d'écritures en section critique, notre objectif a été de minimiser le temps d'attente des tâches sur les lectures ou écritures en section critique. Le principe d'une gestion asynchrone semble alors le moins coûteux comparé à un gestion synchrone.

- La terminaison

La **terminaison générale de l'algorithme** peut se produire de deux manières différentes :

- **A la fin de la phase 2** : le problème peut être :

soit, inchangé : échec

soit, réduit mais non résolu

soit, résolu : toutes les variables sont fixées.

- **Au cours d'une itération de la phase 1** :

soit, l'une des tâches a détecté une contradiction signifiant que la solution optimale du problème est celle associée au minorant
soit, toutes les variables sont fixées.

Toutes les tâches actives en sont informées après le rendez-vous terminant l'itération. Les tâches sont terminées car le problème est résolu.

La *terminaison d'une tâche* peut se produire:

-en phase 1 : par application du test R2 , le nombre de contraintes peut devenir inférieur au nombre de tâches actives, alors que le problème n'est pas terminé .

- en phase 2 : il ne reste plus de contraintes à traiter.

2.2 - Description détaillée et codage

2.2.1 - Variables partagées ou locales ?

L'appellation variable partagée qui apparaît dans cette annexe, désigne une variable commune à plusieurs tâches parallèles, tandis qu'une variable locale désigne une variable réservée strictement à la tâche qui la déclare.

Les variables partagées seront notées en majuscules, pour les différencier des variables locales.

2.2.2 - Définition des variables partagées

Les données du problème:

NP — nombre de processeurs de la machine
NV — nombre de variables du problème au départ.
NC — nombre de contraintes saisies en entrée du programme.
C(1..NC) — tableau des coefficients de la fonction économique telle qu'elle est saisie par le programme en entrée.
A(1..NC,1..NV) — matrice des coefficients du systeme des contraintes du problème.
B(1..NV) — vecteur associé au second membre du système de contraintes
MIN — valeur d'une solution realisable connue

Les variables caractérisant un état du problème :

AR(1..NCA,1..NVA), BR(NCA), CR(NVA)
contiennent les données du probleme réduit, correspondent à A, B, et C.

F0, F1 — signifie qu'au moins une variable a été fixée à 0 ou à 1 au cours de l'itération.
MODIF — une variable a été fixée au cours de l'itération.
MOPT — le minorant est la solution optimale du problème.

X1, X2 — ensemble des variables,
il y a 3 états possibles pour chacune des variables:
-fixée à 0
-fixée à 1
- non fixée.

X1 est modifié au fur et à mesure, X2 mémorise l'état du vecteur à l'itération précédente.

NVA nombre de variables actives.
NVA1 nombre de variables actives à la fin de la phase 1.

NTA nombre de tâches actives.
NT nombre de tâches lancees en parallèle par la tâche initiale
NTA1 nombre total de tâches venues rendez-vous

NCA nombre de contraintes actives
NCA1 nombre de contraintes actives du probleme a la sortie de la phase 1.

TX(1..NP,1..NV) tableau des résultats de chaque tâche :
TX(i,j)=0 ou 1 ou 2
DTX(1..NV) synthèse des résultats enregistrés dans TX(1..NTA,1..NV).

TC(1..NC) tableau indiquant l'état des contraintes:
TC(i)=1 signifie que la contrainte est libre.
TC(i)=2 signifie que la contrainte a été élue par une tâche
TC(i)=0 signifie que la contrainte est éliminée.

TCR(1..NCA)tableau indiquant l'état des contraintes actives en phase 1:

TCR(i)=0 signifie que la contrainte est libre
TCR(i)=1 signifie que la contrainte a été élue par une tâche.

CPT nombre de tâches arrivées au rendez-vous.
NBFINI nombre de tâches arrivées en fin de la phase 2

FIN(1..NP) état des tâches : active ou inactive.

MINOPT(1..NP) détection de terminaison générale par une tâche
MINOPT(i)=vrai si la tâche i a détecté une contradiction.

I_EVT numéro de l'évènement bloquant
I_EVT=0 pendant l'exécution d' une itération d'ordre impair,
I_EVT=1 pendant l'exécution d'une itération d'ordre pair.

EVT(0..1) tableau d'évènements envoyés alternativement lorsque les processus actifs ont terminé une itération de la phase 1.

CPTSEM, FINSEM, TCSEM sémaphores d'exclusion mutuelle sur les variables respectivement: CPT, NBFINI, TC,TCR et NCA.

2.2.3 - Les sections critiques

Les variables: TCR consultée et modifiée par les tâches parallèles en phase 1, TC et NCA, consultées et modifiées par les tâches parallèles en phase 2, mémorisent l'état des contraintes; on dispose du sémaphore TCSEM pour en verrouiller l'accès.

Les compteurs de tâches arrivées en fin de phase 1 (CPT), et en fin de phase 2 (NBFINI) sont les seules autres variables exigeant un traitement en exclusion mutuelle.

2.2.4 - Définition des variables locales

no_ta	numéro de la tâche
iter	numéro du prochain rendez_vous = numéro de l'itération
n	nombre de variables non fixées connu par une tâche
succes1	succés du test R1 (élimination de variables) (phase 1)
succes2	succés du test R2 (élimination de contrainte) (phase 1)
no_evt	numéro de l'evènement attendu par les tâches qui sont arrivées au rendez-vous avant la dernière tâche.

2.2.5 - Définition des procédures utilisées

COND (données: TC(1..NC), TCSEM, i, j; résultat : j) : entier;

-rend le numero de la contrainte j telle que :
j≠i, TC(j) indique que j non éliminée, (i,j) n'a pas déjà été traité par la tâche.
-met à jour j indice de la dernière contrainte utilisée.
-rend 0 si échec dans la recherche

CONTROL(données: TX(1..NV); résultats: MODIF, MOPT, F0, F1)

-détecte une contradiction entre les modifications proposées séparément par les différents processus (MOPT).
-détecte la réduction (MODIF).
-détecte si une variable au moins, a été fixée à 1 (F1) ou à 0 (F0).

ECRES

-écriture sur fichier des résultats du programme:

taille du problème réduit en fin de phase 1 : NCA1, NVA1

taille du problème réduit en fin de phase 2 : NCA, NVA
numéro des contraintes éliminées
numéro, indice et valeur des variables fixées
S'il y a eu réduction complète, valeur de la solution du problème et caractère réalisable ou non de cette solution.

ELIMK (données: k, i, A(1..NC,1..NV), B(1..NC), X1(1..NV): booléen ;

-calcule la valeur du knapsack formé de la première partie de la contrainte k sous la contrainte i, en relachant la condition

d'intégrité sur les variables: $v\overline{(B^k_i)}$

-compare la valeur calculée $v\overline{(B^k_i)}$ avec B(k)
-si cette valeur est inférieure, rend vrai.

FIXVAR (données: no_ta, AR(1..NC,1..NV), BR(1..NC), CR(1..NV), MIN; résultats: minopt, TX, n)

- calcule $v(RB^0{}_i(w_p)\,|\,x_k=\varepsilon)$, pour $(w_p)=(c_p/a_p)$, p=1,..,n et k=1,..n, ε=0 (resp. 1), si $c_{pk} - w_p a_{pk} > 0$ (resp. <0)

-cherche pour chaque variable x_k une valeur qui soit inferieur au minorant choisi pour le problème .

-mémorise dans TX(no_ta) les valeurs des variables éventuellement fixées

-détecte les contradictions telles que: x_i<- 0 et x_i<- 1, le mémorise dans minisbest

-met à jour n, en fonction de ses propres résultats.

LECDON

-effectue la lecture sur fichier des données du problème : A, B, C, MIN
-écrit sur le fichier des résultats, le nom du jeu d'essai et la la valeur du minorant utilisé.

MAJ (données: TX(1..NV), X1(1..NV) ; résultats: DTX(1..NV), TX(1..NP,1..NV), X1(1..NV))

-enregistre les résultats réunis de routes les tâches actives en phase 1 dans le tableau DTX.
-met à jour le vecteur X1, en fonction des résultats produits par les tâches.

-efface TX, prêt pour la prochaine itération.

MAJR1 (donnéees : DTX(1..NV; donnéees-résultats: NVA,NCA, AR(1..NCA,1..NVA), BR(1..NCA), CR(1..NVA))

-met à jour les données du problème réduit par une itération de la phase 1.

MAJR2 (donnéees: NV, NC, X1(1..NV), TC(1..NC), A(1..NC,1..NV), B(1..NC) ; résultats: AR(1..NCA,1..NVA), BR(1..NC)

-met à jour AR et BR (données du problème réduit), à la fin de la phase 2

-met à jour les données du problème réduit par une itération de la phase 1.

NOUVEAU (données: DTX(1..NV),X2(1..NV), TCR(1..NCA); résultat: TCR(1..NCA))

-met à jour TCR pour l'itération suivante, en particulier : détecte les contraintes actives qui n'ont pas été touchées par l'élimination des variables.

NUM_C(données:ir, NC, TC(1..NC)) :entier;

-calcule l'indice de la contrainte ir du problème réduit, par rapport au probleme d'origine.

NUM_R(données: is, X1(1..NV)): entier;

-calcule l'indice de la contrainte is du problème source, par rapport au probleme réduit.

NUM_V(données: ir, NV, X1(1..NV)): entier;

-calcule l'indice de la variable ir dans le problème réduit, par rapport au problème source.

NUM0_V(données: ik, no_c, no_ta, TX(1..NV), X1(1..NV)): entier;

-calcule l'indice de la variable ik du problème de knapsack traité par la tache no_ta, dans le sous programme FIXVAR, par rapport aux données du problème source.

R1 (données: AR(1..NCA,1..NVA), BR(1..NCA); résultats: AR(1..NCA,1..NVA), CR(1..NVA), X1(1..NV), NVA)

-calcule la valeur correspondant à BR(k) - (somme des AR(k,i) tels que x_i<-1) iϵ{1,..,NVA}, kϵ{1..NCA}

-compare chaque coefficient AR(k,i) non nul
-en déduit les variables x_i telles que $x_i<-0$
-inscrit dans le vecteur X1 ses résultats

R2 (données-résultats: AR(1..NVA), BR(1..NCA); résultat: TCR(1..NCA), TC(1..NC), NCA, succes)

-calcule pour kϵ{1..NCA} la valeur correspondant à Σ_i AR(k,i), iϵ{1..NV}.
-compare cette valeur avec BR(k)
-en déduit si la contrainte est redondante
-modifie TC, TCR, NCA, succes si nécessaire

R3 (données: AR(1.. NCA,1..NVA), X1(1..NV); résultats: NVA, X1(1..NV), AR(1.. NCA,1..NVA), CR(1..NVA))

-lit successivement les termes AR(i,j), pour chaque variable x_j, c'est à dire par colonne, tels que la contrainte i ne soit pas éliminée, jusqu'à ce qu'on tombe sur un terme non nul
-si pour une variable x_j , on lit tous les termes AR(i,j) sans en trouver un vérifiant la propriété ci-dessus, alors x_j est fixée à 1.

SOLVE (données:a, b, c, n ; résultat: ib,): réel;

-calcule la valeur du knapsack K (max cx, s.c. ax$\leq$b , x $[0,1]^n$) ϵ continu
-calcule ib, l'indice du multiplicateur w_{ib} tel que

$$v(RK(w_{ib}))=v(K)$$

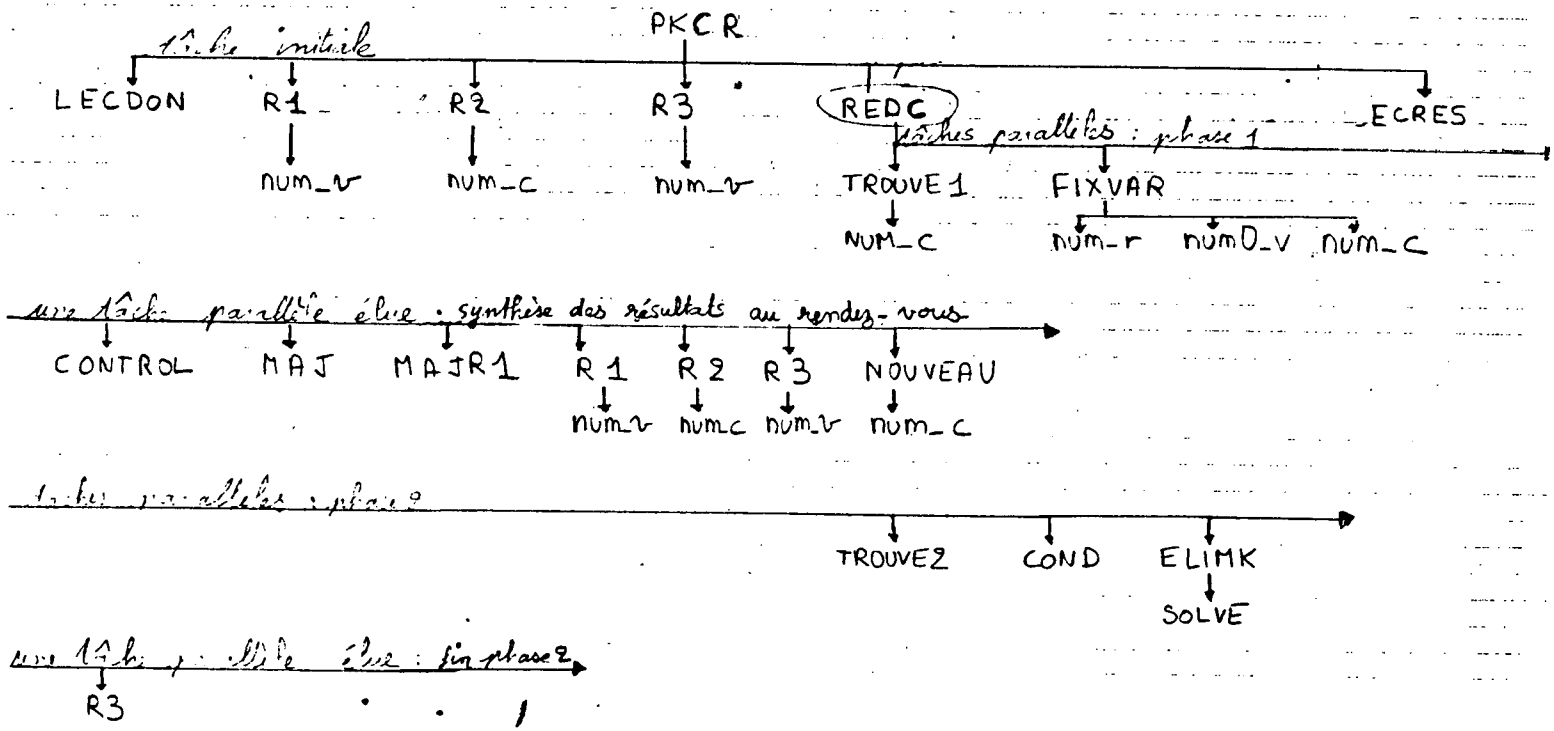
TROUVE1 (données: TCR(1..NCA), TCSEM ; résultat: i)

-cherche un indice i tel que TCR(i) indique que la contrainte i est à traiter en phase 1 : i= contrainte telle que TCR(i)=0;
-change l'état de TCR(i): TCR(i)=1
-utilise le sémaphore TCSEM pour réaliser l'exclusion mutuelle sur l'acces à la variable TCR.

TROUVE2 (données: TC(1..NC), TCSEM ; résultat: i)

-cherche un indice i tel que TC(i) indique que la contrainte i est à traiter en phase 2 : i=contrainte non eliminée;
-change l'état de TC(i):TC(i)=0
-utilise le sémaphore TCSEM pour réaliser l'exclusion mutuelle sur l'acces à la variable TC.

ARBRE DES APPELS



2.2.6 - Initialisation des variables partagées

```
I_EVT=0
EVASGN(EVT(I_EVT);
EVASGN(EVT(1-I_EVT);
EVPOST(EVT(1-I_EVT))

LOCKASGN(FINSEM);
LOCKASGN(TCSEM );
LOCKASGN(CPTSEM );

NCA1:= NC;
NCA:= NC;
NVA:=NV;
NVA1=NV;
CPT:=0; NBFINI:= 0;

Pour i:=1 à NTA faire
          fin(i):= faux;
          MINOPT(i):=faux
Fait

Pour i:=1 à NV faire
          CR(i)=C(i)
          X1(i):=2;
          X2(i):=2;
          TX(i):=2;
          TC(i):=1;
          TCR(i)=0
Fait;
Pour i:=1 à NC faire
          BR(i)=B(i)
          Pour j:=1 à NV faire
          AR(i,j)=A(i,j)
          Fait;
Fait;
MODIF:= faux;
MOPT:= faux;
F0=faux;F1=faux;
```

2.2.7 - Initialisation des variables locales

```
n:=NVA;
succes1:=faux;
succes2=faux;
iter=0
no_evt=0
```

2.2.8 - Algorithme

Tâche initiale

DEBUT

-Lecture des données

-Initialisation des variables partagées

-Réduction selon R1, R2, et R3

-Mise à jour des variables NTA, FIN(1..NP)

-Lancer les tâches "traitement parallèle"

-Exécuter "traitement parallèle"

-Attendre fin de toutes les tâches lancées en parrallèle

FIN.

Tâche "traitement parallèle"

Paramètre transmis : no_ta, le numéro de la tâche

DEBUT

Tantque non(MOPT ou NVA=0 ou FIN(no_ta)) et MODIF faire

```
n:= NVA;
Tantque non MINOPT(no_ta) et TROUVE1( TCR, TCSEM, i) et n≠0 faire
   FIXVAR(i, AR, CR, BR, MIN, TX(no_ta), MINISBEST(no_ta), n);
Ftque;

P(CPTSEM)
CPT:=CPT + 1;
no_evt=I_EVT
```

```
Si CPT≠NTA
Alors V(CPTSEM);
        EVWAIT(EVT(no_evt))
Sinon V(CPTSEM)
        NTA1=NTA

        Si non MOPT
        Alors   CONTROL(TX, MODIF, MOPT, F0, F1);
                Si non MOPT et MODIF
                Alors   MAJ(TX,DTX, X1, NVA);
                        MAJR1(DTX, AR,BR,CR,NVA)
                        Si NVA ≠0
                        Alors succes1=faux;
                                Si F1
                             Alors R1(AR, BR, X1, NVA,succes1)
                                Fsi;
                                Si (NVA≠0 et (F0 ou succes1))
                                Alors succes2=faux;
                                        R2(A, B, X1, TC, NCA,succes2)
                                        Si succes2
                                        Alors
R3(A, X1, TC, NVA);
Si NVA≠0 et NCA<NP
Alors Pour i:=NCA+1 à NTA faire
         FIN(i):=vrai
      Fait;
      NTA:=NCA
Fsi
                                        Fsi
                                Fsi
                        Fsi;

                        Si non MOPT et NVA≠0
                        Alors   Si MODIF
                                Alors   NOUVEAU(X1, X2, A, TC);
                                        X2:=X1
                                Fsi
                        Fsi
                Fsi
        Fsi;

      NCA1=NCA;
      NVA1=NVA;
      CPT=0
      I_EVT=1-I_EVT
      EVCLEAR(EVT(I_EVT))
      EVPOST(EVT(1-I_EVT))
  Fsi;
Ftque;
```

```
Si non ( MOPT ou NVA=0 ou NCA=1 ou FIN(no_ta))
Alors   i:=1;
        Tantque TROUVE2(TC, TCSEM, i) ≤ NC faire
           j:=1;

           Tantque COND(TC, TCSEM, i, j) ≤ NC faire

              Si ELIMK(i, j, A, B)
              Alors   P(TCSEM)
                      Si TC(j)≠0
                      Alors TC(i)=0
                            NCA=NCA -1
                      Fsi
                      V(TCSEM)
              Fsi

           Ftque
        Ftque;
     P(FINSEM)
     NBFINI := NBFINI + 1
     Si NBFINI = NTA et  NCA≠0 et NCA ≠ NCA1
     Alors MAJR2(A, X1, TC, AR)
           R3(AR,NVA,X1)
     Fsi
     V(FINSEM)
Fsi

FIN
```

2.3 - Choix effectués pour le codage en Fortran 77

2.3.1 - **Pour une version unique**

Le compilateur Fortran 77, tel qu'il existe actuellement sur le CRAY 2, permet de déclarer des variables communes aux differentes tâches lancées dans un même programme - instruction COMMON . Par contre, on ne peut pas déclarer des variables communes à des sous-programmes appelés par des tâches différentes, sans que celles ci ne soient nécessairement partagées par ces tâches. L'instruction "Task Common", qui existe a cet effet n'est pas en service.

Le logiciel CREM, qui a servi à mettre au point le programme, permet de déclarer: des variables partagées entre les tâches -instruction COMMON$ -, et des variables locales à une tâche, partagées par des sous-programmes appelés -instruction COMMON -.

Nous nous sommes efforcés de développer une version unique du programme PKCR, dans les limites des contraintes logicielles, telles que celle évoquée ci-dessus.

2.3.2 - **Limiter le nombre de paramètres des sous-programmes**

Le passage de paramètres est plus rapide par l'instruction COMMON.

Pour les variables partagées, on peut les déclarer dans un COMMON, dans le sous-programme appelé et appelant, par contre, on est obligé de transmettre les variables locales, comme paramètres du sousprogramme appelé.

2.3.3 - **Optimisation du code**

Pour accélérer l'exécution du code FORTRAN, nous nous sommes efforcés de désemboiter les boucles imbriquées, de placer les boucles les plus longues au niveau le plus bas, et de faire appel à des fonctions optimisées de la bibliothèque $SCILIB.

Parmi les fonctions optimisées de la bibliothèque $SCILIB, nous avons surtout utilisé les suivantes:

ISEARCH(n,vecteur,inc,objet) fonction de type entier, donne l'indice du premier élément du vecteur ayant pour valeur objet.

SAXPY(N,SA,SX,INCX,SY,INCY) calcule Y = AX + Y

Chaque fois que le nombre d'itérations d'une boucle est suffisant, il devient intéressant de la vectoriser.

La séquence : *si (var.op.expression) var=expression* empêche la vectorisation d'une boucle. Elle peut être optimisée par une instruction de la forme *var=function(var,expression)*, où *function* est une version des fonctions MAX/MIN. Ainsi, on ramène une boucle non vectorisable à une boucle vectorisable.

Pour les instructions de type *si(expression logique) var=expression*, nous avons employé la fonction **CVMGT**: *var=CVMGT(expression,var,condition)*.

3. LISTING DU PROGRAMME PR²

```
**********************************************************************
*                                                                    *
*             Bienvenue sur le CRAY-2 du CCVR                        *
*                                                                    *
**********************************************************************
No mail.
news: assist.June10 com.June09 planning plncar.Mai20 disk_sp.May24
trap 'jar -t'

cd ft

rm -f pkr.f

jad

cat <<'EOF' >pkcr.f
      program pkcr

c parallel knapsack reduction

      implicit integer(a-z)

c definition des constantes

      parameter(np=4)
      parameter (nvmax=199,ncmax=101)
      parameter(res=11)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_glob/f0,f1,modif,mopt
      common/e_proc/fin(np),tx(np,nvmax),minopt(np),nta,nta1
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)
      common/e_prob/ar(ncmax,nvmax),br(ncmax),cr(nvmax)
      common/sync1/cpt,evt(0:1),cptsem,i_evt
      common/sync2/nbfini,finsem,tcsem
      dimension tca(3,np)
      logical fin,f0,f1,modif,mopt,minopt,succes1,succes2
      external redc

c lecture des donnees

      call lecdon


c lecture du nombre de taches

      write(res,*)"nombre de taches?"
      read(*,*)nta
      write(res,*)nta

c initialisation des variables partagees

      do 22 i=1,nc
          br(i)=b(i)
22    continue
      do 23 i=1,nv
          cr(i)=c(i)
23    continue
      do 24 i=1,nc
          do 241 j=1,nv
                    ar(i,j)=a(i,j)
241       continue
24    continue
```

```
      f0=.false.
      f1=.false.
      modif=.false.
      mopt=.false.
      cpt=0
      nbfini=0
      nva=nv
      nta=min0(nta,nc)
      nca=nc
      nca1=nc

      do 8 j=1,nv
          x1(j)=2
          x2(j)=2
8         continue
      do 81 j=1,nv
          do 7 i=1,nta
                    tx(i,j)=2
7        continue
81    continue
      do 9 i=1,nc
          tc(i)=1
          tcr(i)=0
9     continue

      do 13 i=1,nta
          fin(i)=.false.
          minopt(i)=.false.
13    continue

c reduction selon r1,r2,r3

      if(nv.gt.0)call r1(res+1,succes1)
      if(nc.gt.0)call r2(res+1,succes2)
      if(nva.gt.0)call r3(res+1)
      if(nva.ne.nv)then
          do 15 i=1,nv
                    x2(i)=x1(i)
15        continue
      endif

c mise a jour de nta et fin(np)

      if(nca.lt.nta)then
          nta=nca
          do 20 i=nta,nca+1,-1
                    fin(i)=.true.
20        continue
      endif
      nta1=nta

c initialisation des evenements et des verrous

      if(nta.gt.1)then
          call lockasgn(finsem)
          call lockasgn(tcsem)
          call lockasgn(cptsem)
          call evasgn(evt(0))
          call evasgn(evt(1))
          i_evt=0
          call evpost(evt(1))

c activation des taches paralleles
          nt=nta
          do 21 i=1,nta-1
                    tca(1,i)=3
```

```
                    tca(3,i)=i
                    call tskstart(tca(1,i),redc)
21        continue
      endif

      call redc

c attente de fin d"execution des taches paralleles

      if(nt.gt.1)then
          do 25 i=1,nt-1
                    call tskwait(tca(1,i))
25        continue
      endif
c ecriture des resultats

      call ecres
      stop
      end
c**********************************************************************
      subroutine redc

c traitement effectue par une tache
c gestion des conflits d"acces aux variables partagees :nbfini,cpt,tc,nca
c avec les verrous :finsem, cptsem et tcsem
c gestion d"un rv avec un evenement : evt
c traitement comportant :2 phases distinctes et successives
c - phase 1: pour fixer les variables
c - phase 2: pour eliminer des contraintes
c nb : les taches sont synchronisees apres la phase 1

      implicit integer(a-z)


c definition des constantes

      parameter(np=4)
      parameter (nvmax=199,ncmax=101)
      parameter(res=11)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_glob/f0,f1,modif,mopt
      common/e_proc/fin(np),tx(np,nvmax),minopt(np),nta,nta1
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)
      common/sync1/cpt,evt(0:1),cptsem,i_evt
      common/sync2/nbfini,finsem,tcsem

      logical fin,f0,f1,modif,mopt,minopt,succes1,succes2
      logical elimk

      call tskvalue(no_ta)
      no_ta =no_ta+1
      u=res+no_ta
      write(u,*)"no_ta=",no_ta
      iter=0
      write(u,*)"phase 1"
      go to 1
10    continue
      if (mopt.or.nva.eq.0.or.fin(no_ta).or..not.modif) go to 101
1     continue
      iter=iter+1

c phase 1:fixer des variables
```

```
      n=nva
      i=0
100   continue
      if(minopt(no_ta))go to 111
      if(n.eq.0)go to 111
      if(i.ge.nca)go to 111
      i=trouve1(i,u)
      if(i.eq.0)go to 111
      call fixvar(no_ta,i,n)
      go to 100

c fin des calculs individuels : rendez-vous

111   continue
      write(u,*)"rendez-vous  ",iter

      if (nta.gt.1)then
          call lockon(cptsem)
          cpt=cpt+1
          no_evt=i_evt

c attente sur l"envoi de l"evt indiquant la fin du travail de synchro,
c travail execute par la derniere tache arrivee au r.v.
          if(cpt.ne.nta)then
                    call lockoff(cptsem)
                    call evwait(evt(no_evt))
                    go to 10
          endif
          call lockoff(cptsem)
          nta1=nta
      endif

c collecte des resultats obtenus par les taches actives
c la tache effectuant la collecte est la derniere tache qui est arrivee
c au rendez-vous


      if (isearch(nta,minopt(1),1,.true.).le.nta)mopt=.true.
      if (.not.mopt)then
          write(u,*)"non mopt au 1er controle"
          call control
          if(.not.mopt)then
                    write(u,*)"non mopt au 2eme controle"
                    if (modif)then
                              write(u,*)"modif=vrai"
                              call maj
                              call majr1
                              succes1=.false.
                              if(f1.and.nva.ne.0)call r1(u,succes1)
                              if ((f0.or.succes1).and.nva.ne.0)then
                                        succes2=.false.
                                        call r2(u,succes2)
                                        if(succes2) call r3(u)

c desactivation de certaines taches si le nombre de contraintes est
c devenu inferieur au nombre de contraintes du probleme reduit

                                        if (nva.ne.0.and.nca.lt.nta)
     *then
          do 13 i=nta,nca+1,-1
                    fin(i)=.true.
13        continue
          nta=nca
                                        endif
                              endif
                              if(nva.ne.0)then
```

```
                                        call nouveau
                                        do 30 i=1,nv
                                                x2(i)=x1(i)
30                                      continue
                                endif

c probleme inchange, pret a passer en phase 2

                        else
                                write(u,*)"modif=faux"
                        endif
                else
                        write(u,*)"mopt=vrai au 2eme controle"
                endif
        else
                write(u,*)"mopt=vrai au 1er controle"
        endif

        nca1=nca
        nva1=nva
c fin de la collecte des resultats des differentes taches actives

        if(nta1.gt.1)then

c mise a jour des variables de synchronisation
c liberation des taches en attente au rendez-vous

                cpt=0
                i_evt=1-i_evt
                call evclear(evt(i_evt))
                call evpost(evt(1-i_evt))
        endif
        go to 10
101     continue
c fin de l"elimination des variables

c phase 2: eliminer des contraintes

        write(u,*)"phase 2"
        if (.not.(mopt.or.nca.eq.1.or.nva.eq.0.or.fin(no_ta)))then
                i=0
20              continue

c selection d"une contrainte libre
                if(i.ge.nc) go to 201
                i=trouve2(i)
                if (i.eq.0) go to 201
                j=0
21              continue

c selection d"une autre contrainte active

                j=cond(i,j)
                if(j.eq.0)go to 20
                if (elimk(i,j,u))then
                        write(u,*)"elimk(",i,"," ,j, ")=vrai"
                        if(nta.gt.1)call lockon(tcsem)
                        if(tc(j).ne.0)then
                                tc(i)=0
                                nca=nca-1
                                if(nta.gt.1) call lockoff(tcsem)
                                go to 20
                        else
                write(u,*)j," eliminee par une autre tache: ",i," conservee"
                                if(nta.gt.1) call lockoff(tcsem)
                                go to 21
```

```
                endif


c          else
c                   write(u,*)"elimk(",i, "," ,j, ")=faux"

          endif
          go to 21

201       continue
          write(u,*)"phase 2 terminee"

c fin des tests pour eliminer des contraintes
c fin de la tache, si la tache est la derniere a se terminer alors:
c fin du programme : appliquer r3 si besoin.

          if (nta.gt.1) call lockon(finsem)
          nbfini =nbfini+1
          if(nbfini.eq.nta)then
                    if (nca.ne.nca1)then
                              write(u,*)"r3"
                              if(nca.gt.0)call majr2
                              call r3(u)
                    endif
          endif
          if (nta.gt.1) call lockoff(finsem)
     endif

     write(u,*)"tache terminee"

     return
     end

c*********************************************************************
     subroutine fixvar(no_ta,no_c,n)

c calcule une matrice de majorants de la valeur du knapsack
c extrait du probleme source, en deduit les variables qui pourront
c etre fixees


     implicit integer(a-z)

c definition des constantes
     parameter(res=11)
     parameter(np=4)
     parameter (nvmax=199,ncmax=101)

c definition des variables partagees

     common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
     common/e_prob/ar(ncmax,nvmax),br(ncmax),cr(nvmax)
     common/e_proc/fin(np),tx(np,nvmax),minopt(np),nta,nta1
     common/e_var/nva,nva1,x1(nvmax),x2(nvmax)
     common/saisie/min

c       common/y/ y
     logical fin,minopt,xk(2,nvmax)
     dimension ck(nvmax),ak(nvmax),mj(nvmax,nvmax)
     real y(nvmax,nvmax),lambda(nvmax),r(nvmax)

c initialiser les variables locales

     u=res+no_ta

     do 4 i=1,nva
```

```
          ck(i)=cr(i)
          ak(i)=ar(no_c,i)
4     continue

      bk=br(no_c)
      nk=nva

      do 6 j=nv,1,-1
      if(x1(j).eq.2)then
          jr=num_r(j)
          if (tx(no_ta,j).ne.2.or.ak(jr).eq.0)then
                    if(tx(no_ta,j).eq.1)bk=bk-ak(jr)
                    do 7 i=jr,nk
                              ak(i)=ak(i+1)
                              ck(i)=ck(i+1)
7                   continue
                    nk=nk-1
          endif
      endif
6     continue
      if(nk.eq.0)return

      do 1 i=1,nk
          do 2 j=1,nk
                    y(i,j)=0.
                    mj(i,j)=0
2         continue
1     continue
      do 3 i=1,nk
          xk(1,i)=.false.
          xk(2,i)=.false.
          lambda(i)=0.
          r(i)=0.
3     continue


c tests r1 et r2 appliques au knapsack

      if(n.lt.nva)then
          do 24 j=nk,1,-1
                    if(ak(j).gt.bk)then
                              j0=num0_v(j,no_c,no_ta)
                              tx(no_ta,j0)=0
                              write(u,*)"x",j0,"<— 0"
                              n=n-1
                              nk=nk-1
                              do 22 i=j,nk
                                        ck(i)=ck(i+1)
                                        ak(i)=ak(i+1)
22                            continue
                    endif
24        continue
          if(nk.eq.0)return
      s=0
      do 23 j=1,nk
          s=s+ak(j)
23    continue
      if(s.le.bk)return

       endif
c calculer la matrice des majorants de la valeur du knapsack

      s=0
      do 20 j=1,nv
          if(x1(j).eq.1)then
                    s=s+c(j)
```

```
        else
                if(tx(no_ta,j).eq.1)then
                        s=s+c(j)
                else
                        if(a(num_c(no_c),j).eq.0)s=s+c(j)
                endif
        endif
20    continue
      do 21 i=1,nk
            r(i)=real(s)
21    continue

      do 81 i=1,nk
        lambda(i)=real(ck(i))/real(ak(i))
81    continue
      call saxpy(nk,real(bk),lambda,1,r,1)
      do 8 i=1,nk
        do 9 j=1,nk
                y(i,j)=real(ck(j))-lambda(i)*real(ak(j))
                r(i)=r(i)+amax1(y(i,j),0.0)
9       continue

        do 10 j=1,nk
                mj(i,j)=max0( int(r(i) - abs(y(i,j)))-min ,0)
10      continue
8     continue


c     lire mj, remplir xk et minopt


       do 11 j=1,nk
        i=0
14      continue
        i=i+1
        i=i+isearch(nk+1-i,mj(i,j),1,0)-1
        if(i.le.nk)then
                if(y(i,j).gt.0)xk(2,j)=.true.
                if(y(i,j).lt.0)xk(1,j)=.true.
        if(i.lt.nk.and..not.(xk(1,j).and.xk(2,j)))go to 14
        if(xk(1,j).and.xk(2,j))minopt(no_ta)=.true.
        if(minopt(no_ta))return
        endif

11    continue


c collecte des resultats: lecture de xk, remplir tx(no_ta,*)

      do 13 j=nk,1,-1
        if (xk(1,j))then
                j0=num0_v(j,no_c,no_ta)
                tx(no_ta,j0)=0
                write(u,*)"x",j0,"<-- 0"
                n=n-1
        else
                if(xk(2,j))then
                        j0=num0_v(j,no_c,no_ta)
                        tx(no_ta,j0)=1
                        n=n-1
                        write(u,*)"x",j0,"<-- 1"

                endif
        endif
13    continue
```

```
      return
      end


c*****************************************************************
      subroutine r1(u,succes)

c detecte les variables qui peuvent etre fixees a 0
c enregistre les resultats, nva x1
c modifie les donnees du probleme reduit

      implicit integer(a-z)

c definition des constantes

      parameter (nvmax=199,ncmax=101)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)
      common/e_prob/ar(ncmax,nvmax),br(ncmax),cr(nvmax)
      logical succes

c code optimise pour etre vectorise

      k=0
      do 20 j=1,nca
          do 21 i=nva,1,-1
                    if (ar(j,i).gt.br(j))then
                              succes=.true.
                              nva=nva-1
                              i0=num_v(i)
                              x1(i0)=0
                              write(u,*)"R1 : x",i0,"<-0"
                              do 22 k=i,nva
                                        do 231 l=1,nca
                                                  ar(l,k)=ar(l,k+1)
231                                       continue
                                        cr(k)=cr(k+1)
22                            continue
                    endif
21        continue
20    continue

      return
      end
c*****************************************************************
      subroutine r2(u,succes)

c detecte les contraintes redondantes
c enregistre les resultats: nca, tc, tcr, succes
c modifie les donnees du probleme reduit : ar,tar,br

      implicit integer(a-z)

c definition des constantes

      parameter (nvmax=199,ncmax=101)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)
```

```
      common/e_prob/ar(ncmax,nvmax),br(ncmax),cr(nvmax)

      dimension s(ncmax)
      logical succes


      do 19 j=1,nca
          s(j)=0
19     continue

      do 20 j=nca,1,-1
          do 210 i=1,nva
                    s(j)=s(j) + ar(j,i)
210       continue
          if (s(j).le.br(j))then
                    nca=nca-1
                    succes=.true.
                    j0=num_c(j)
                    tc(j0)=0
                    write(u,*)"R2 : tc(",j0,")=0"
                    do 21 k=j,nca
                              tcr(k)=tcr(k+1)
                              br(k)=br(k+1)
                              do 222 i=1,nva
                                        ar(k,i)=ar(k+1,i)
222                           continue
21                  continue
          endif
20     continue

      return
      end
c*************************************************
      subroutine r3(u)

c detecte les variables a fixer a 1
c met a jour: nva, x1
c modifie les donnees du probleme reduit : ar,tar,cr

      implicit integer(a-z)

c definition des constantes

      parameter (nvmax=199,ncmax=101)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)
      common/e_prob/ar(ncmax,nvmax),br(ncmax),cr(nvmax)

      if (nca.eq.0)then
          do 30 k=1,nv
                    x1(k)=min0(1,x1(k))
30        continue
          nva=0
          return
      endif

      do 20 j=nva,1,-1
          i=0
21        i=i+1
          if(ar(i,j).eq.0)then
                    if(i.lt.nca)go to 21
                    if(i.eq.nca)then
```

```
                              nva=nva-1
                              j0=num_v(j)
                              x1(j0)=1
                              write(u,*)"R3 : x",j0,"<-1"
                              do 212 k=j,nva
                                        cr(k)=cr(k+1)
                                        do 221 l=1,nca
                                                  ar(l,k)=ar(l,k+1)
221                                       continue
212                           continue
                    endif
          endif
20    continue

      return
      end
c*****************************************
      subroutine ecres

c ecriture des resultats sur un fichier de sortie

      implicit integer (a-z)

c definition des constantes

      parameter (nvmax=199, ncmax=101)
      parameter (res=11)

c definition des variables

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_glob/f0,f1,modif,mopt
      common/saisie/min
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)


      dimension xx0(nvmax),xx1(nvmax),ce(ncmax)
      logical f0,f1,modif,mopt


c ecriture des resultats

      write(res,99) nc,nv
99    format("taille du probleme initial: ",i3,i4)

c probleme pourlequel la valeur de la solution donnee
c comme minorant de la valeur optimale du probleme est optimale

      if (mopt) then
          write(res,106) min
          write(res,100) nca,nva

      else

          if(nva.eq.0)nca=0
          write(res,15) nca1,nva1
15        format("phase 1: taille du probleme reduit: ",i3,i4)
          write(res,100) nca,nva
100       format("phase 2: taille du probleme reduit: ",i3,i4)


c probleme avec des contraintes eliminees

      if  (nva.ne.0.and.nca.lt.nc) then
          write(res,101)
```

```
101       format("numero des contraintes eliminees: ")
          nbce=0
          do 10 i=1,nc
                    if (tc(i).eq.0)then
                              nbce=nbce+1
                              ce(nbce)=i
                    endif
10        continue
          write(res,102) (ce(i),i=1,nbce)
102       format(30i3)
          •
      endif

c probleme avec des variables fixees

      if (nva.lt.nv) then
          nv0=0
          nv1=0

          do 11 i=1,nv
                    if (x1(i).eq.0) then
                              nv0=nv0+1
                              xx0(nv0)=i

                    else
                                        if(x1(i).eq.1) then
                                            nv1=nv1+1
                                            xx1(nv1)=i
                                        endif
                    endif
11        continue
          if (nv0.ne.0)then
                    write(res,103)
103                 format("numero des variables fixees a 0: ")
                    write(res,104)(xx0(i),i=1,nv0)
104                 format(15i4)
          endif
          if (nv1.ne.0)then
                    write(res,105)
105                 format("numero des variables fixees a 1: ")
                    write(res,104)(xx1(i),i=1,nv1)
          endif

c probleme resolu

          if (nva.eq.0) then
                    v=0
                    do 12 i=1,nv
                              v=v+ (c(i)*x1(i))
12                  continue
                    write(res,106) v
106                 format("valeur optimale du probleme: ", i7)

                    j=0
14                  continue
                    j=j+1
                    v=0
                    do 13 i=1,nv
                              v=v + (a(j,i)*x1(i))
13                  continue
                    if (v.gt.b(j))then
                              write(res,107)
107                           format("solution non realisable")
                    else
                              if (j.lt.nc)go to 14
                              write(res,108)
```

```
108                     format("solution realisable")
                  endif


         endif

      endif
      endif

90    continue
      return
      end
c*************************************************************
      subroutine lecdon

c lecture des donnees du probleme de multiknapsack
c A, B, C, NV, NC

      implicit integer (a-z)

c definition des constantes
      parameter(don=10)
      parameter(nvmax=199, ncmax=101)
      parameter(res=11)

c definition des variables
      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/saisie/min

      character*8 titre
      dimension bb(ncmax)

      write(res,*)"probleme test: "
      read(*,101)titre
      write(res,101)titre
101   format(a8)
c choix du sous probleme

      read(don,*) nbp, nc

      noselect=1
      if (nbp.gt.1) then
30                write(res,10)nbp
10    format('selectionner un numero parmi les ',i2," problemes")
                  read(*,*)noselect
                  write(res,*)noselect
      endif

c lecture de la valeur du minorant

      write(res,*)"valeur du minorant"
      read(*,*)min
      write(res,*)min

c lecture des donnees a, b, c

      do 32 n=1,noselect
32       read(don,100) nv, (b(i), i=1,nc)


      do 33 n=1,nbp-noselect
33       read(don,100) nvb, (bb(i), i=1,nc)

      do 34 j=1,nv
34       read(don,100,end=105) c(j), (a(i,j), i=1,nc)
```

```
100   format(10i6)
105  continue
     return
     end
c*************************************************************
     subroutine control

c met a jour mopt,f0,f1,modif, en fonction des resultats obtenus
c par les differentes taches

     implicit integer(a-z)

c definition des constantes

     parameter(np=4)
     parameter (nvmax=199,ncmax=101)

c definition des variables partagees

     common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
     common/e_glob/f0,f1,modif,mopt
     common/e_proc/fin(np),tx(np,nvmax),minopt(np),nta,nta1
     common/e_var/nva,nva1,x1(nvmax),x2(nvmax)

     logical fin,f0,f1,modif,mopt,minopt,v0,v1

c initialisation des variables de controle

     f0=.false.
     f1=.false.
     modif=.false.
     mopt=.false.

c calcul de mopt

     j=1
10   continue
     j=j+isearch(nv+1-j,x1(j),1,2)-1
     if(j.gt.nv)go to 12
     i1=isearch(nta,tx(1,j),1,1)
     i0=isearch(nta,tx(1,j),1,0)
     if(i1.le.nta.and.i0.le.nta)then
         mopt=.true.
         return
     endif
     if(j.lt.nv)then
         j=j+1
         go to 10
     endif
12   continue

c fin du calcul de mopt

c calcul de f0
     j=0
20   continue
     j=j+1
     if (isearch(nv,tx(j,1),4,0).le.nv)f0=.true.
     if(j.lt.nta.and..not.f0)go to 20

c calcul de f1
     j=0
21   continue
     j=j+1
     if (isearch(nv,tx(j,1),4,1).le.nv)f1=.true.
```

```
      if(j.lt.nta.and..not.f1)go to 21

c calcul de modif

      if(f0.or.f1)modif=.true.
      return
      end


c************************************************
      function num0_v(ik,j,no_t)

c transforme ik dans[1,nk]  en i dans [1,nv]

      implicit integer(a-z)

c definition des constantes

      parameter (nvmax=199,ncmax=101)
      parameter (np=4)

c definition des variables partagees

      common/e_prob/ar(ncmax,nvmax),br(ncmax),cr(nvmax)
      common/e_proc/fin(np),tx(np,nvmax),minopt(np),nta,nta1
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)
      logical fin,minopt

      i=0
      do 20 n=1,ik
10        continue
          i=i+1
          if(x1(i).lt.2)go to 10
          if(tx(no_t,i).lt.2)go to 10
          if(ar(j,num_r(i)).eq.0)go to 10
20    continue

      num0_v=i

      return
      end
c*******************************************************
      function num_c(ir)

c donne l"indice correspondant a ir dans br (nca<nc), dans le tc (nc)

      implicit integer(a-z)

c definition des constantes

      parameter (ncmax=101,nvmax=199)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)

      is=0
      do 20 i=1,ir
10    continue
      is=is+1
      if (tc(is).eq.0)go to 10
c       is=is+isearch(nc+1-is,tc(is),1,1)-1
20    continue
      num_c=is
      return
```

```
      end

c     ==========================================
c
      function num_v(ir)

c donne l"indice correspondant a ir dans l"intervalle [1,nva],
c dans [1,nv]

      implicit integer(a-z)

c definition des constantes

      parameter (nvmax=199,ncmax=101)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)

      is=0
      do 1 i=1,ir
          is=is+1
          is=is+isearch(nv+1-is,x1(is),1,2)-1
1     continue
      num_v=is
      return
      end
c     ==========================================


      function num_r(is)

c effectue le changement de base entre l"indice  is, variant sur
c [1,nv] et l"indice ir, variant sur [1,nva]

      implicit integer(a-z)

c definition des constantes

      parameter (nvmax=199)

c definition des variables partagees

      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)

      ir=is
      do 1 i=1,is
          ir=cvmgt(ir,ir-1,x1(i).eq.2)
1     continue
      num_r=ir
      return
      end

c*************************************************************

      function trouve1(i1,u)

c rend le numero de la contrainte libre choisie pour la phase1
c rend 0 si il n"y a plus de contraintes libres
c change l"e de la contrainte selectionnee : libre__> reservee

      implicit integer(a-z)

c definition des constantes
```

```
      parameter(np=4)
      parameter (nvmax=199,ncmax=101)
      parameter(res=11)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_proc/fin(np),tx(np,nvmax),minopt(np),nta,nta1
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/sync2/nbfini,finsem,tcsem

      i1=i1+1
      if(nta.gt.1)call lockon(tcsem)
      i1=i1+isearch(nca+1-i1,tcr(i1),1,0)-1
3     if(i1.gt.nca)then
          trouve1=0
      else
                    tcr(i1)=1
                    trouve1=i1
                    write(u,*)"i=",num_c(i1)
      endif

      if(nta.gt.1)call lockoff(tcsem)

      return
      end
c     ==========================================

      subroutine majr1

c met a jour ar,tar,br,cr en fonction de dtx

       implicit integer(a-z)

c definition des constantes

      parameter (nvmax=199,ncmax=101)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_prob/ar(ncmax,nvmax),br(ncmax),cr(nvmax)
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)
      common/maj_var/dtx(nvmax)

      dimension dtxr(nvmax)

c reduire dtx pour l"ajuster avec les donnees du pb reduit de l"iteration
c precedente
      do 12 j=1,nv
          dtxr(j)=dtx(j)
12    continue
      do 13 j=nv-1,1,-1
          if(x2(j).ne.2)then
                    do 14 k=j,nv-1
                              dtxr(k)=dtxr(k+1)
14                  continue
          endif
13    continue



c decalage a gauche de ar,cr, vers le haut de br
c modification des valeurs contenues dans br
```

```
      do 10 j=nva,1,-1
          if (dtxr(j).ne.2)then
                    nva=nva-1
                    do 22 k=j,nva
                              cr(k)=cr(k+1)
22                  continue
                    if(dtxr(j).eq.1)then
                              do 20 i=1,nca
                                        br(i)=br(i)-ar(i,j)
20                            continue
                    endif
                    do 23 k=j,nva
                              do 21 i=1,nca
                                        ar(i,k)=ar(i,k+1)
21                            continue
23                  continue
          endif
10    continue

      return
      end
c     ==========================================
c
      subroutine maj

c calcule la nouvelle valeur de x1 a partir des resultats des
c differentes taches

c resume tx _2 dimensions_, dans dtx _1 dimension_
c efface le tableau tx : pret pour la prochaine iteration


      implicit integer(a-z)

c definition des constantes

      parameter(np=4)
      parameter (nvmax=199,ncmax=101)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_proc/fin(np),tx(np,nvmax),minopt(np),nta,nta1
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)
      common/maj_var/dtx(nvmax)

c initialiser dtx

      do 15 i=1,nv
          dtx(i)=2
15    continue
      do 11 j=1,nv
          i1=isearch(nta,tx(1,j),1,1)
          if(i1.le.nta)then
                              x1(j)=1
                              dtx(j)=1
          else
                              i0=isearch(nta,tx(1,j),1,0)
                              if(i0.le.nta)then
                                        x1(j)=0
                                        dtx(j)=0
                              endif
          endif
11    continue

c effacer tx
```

```
      do 12 i=1,nta
          do 13 j=1,nv
                    tx(i,j)=2
13        continue
12    continue

      return
      end
c     ==========================================

      subroutine nouveau

c prepare le tableau tcr pour l"iteration suivante

      implicit integer(a-z)

c definition des constantes

      parameter (nvmax=199,ncmax=101)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/maj_var/dtx(nvmax)

      do 1 i=1,nca
          tcr(i)=1
1     continue

      do 10 j=1,nv
          if(dtx(j).ne.2)then
                    do 5 i=1,nca
                              if(tcr(i).eq.1)then
                                        if(a(num_c(i),j).ne.0)then
                                                  tcr(i)=0
                                        endif
                              endif
5                   continue
          endif
10    continue

      return
      end
c     ==========================================
c
      function cond(c1,c2)

c c1 = numero de la contrainte a eliminer
c c2 = numero de la derniere contrainte selectionnee par cond

c rend  le numero de la contrainte elue ou bien 0 si toutes
c les contraintes actives ont deja servi

      implicit integer(a-z)

c definition des constantes

      parameter(np=4)
      parameter (nvmax=199,ncmax=101)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_proc/fin(np),tx(np,nvmax),minopt(np),nta,nta1
```

```
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/sync2/nbfini,finsem,tcsem

      c2=c2+1
      if(nta.gt.1)call lockon(tcsem)
20        continue
          if(c2.eq.c1)c2=c2+1
          if (c2.gt.nc)then
                    cond=0
          else
                    if (tc(c2).ne.0)then
                              cond=c2
                    else
                              c2=c2+1
                              go to 20
                    endif
          endif
      if(nta.gt.1)call lockoff(tcsem)

      return
      end
c*************************************************************

      function trouve2(i1)

c rend le numero de la contrainte libre choisie pour la phase2
c rend 0 si il n"y a plus de contraintes libres
c change l"e de la contrainte selectionnee : libre__> reservee

      implicit integer(a-z)

c definition des constantes

      parameter(np=4)
      parameter (nvmax=199,ncmax=101)

c definition des variables partagees

      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      common/e_proc/fin(np),tx(np,nvmax),minopt(np),nta,nta1
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/sync2/nbfini,finsem,tcsem

      if(nta.gt.1)call lockon(tcsem)
c       i1=i1+isearch(nc+1-i1,tc(i1),1,1)-1

20    continue
      i1=i1+1
      if(i1.gt.nc)then
            trouve2=0
      else
          if (tc(i1).eq.1)then
              tc(i1)=2
              trouve2=i1
          else
                    go to 20
          endif
      endif
      if(nta.gt.1)call lockoff(tcsem)

      return
      end
c*************************************************************
      logical function elimk(i,j,u)

c applique un test sur la contrainte i, en utilisant la contrainte j
```

```
c enregistre le resultat du test dans la variable succes

      implicit integer(a-z)

c definition des constantes

      parameter (nvmax=199,ncmax=101)

c definition des variables partagees

      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)
      common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
      dimension ai(nvmax),aj(nvmax)
      logical test

c initialiser les variables definissant le knapsack :
c max ai*x, s.c. aj*x<b ou aj*x=b
      do 10 k=1,nv
          ai(k)=a(i,k)
          aj(k)=a(j,k)
10       continue
      bj=b(j)
      bi=b(i)
      lv=nv
      do 11 k=nv,1,-1
          if(ai(k).eq.0.or.aj(k).eq.0.or.x1(k).lt.2)then
                    if(ai(k).eq.0.and.aj(k).ne.0)then
                              elimk=.false.
                              return
                    endif
                    if(aj(k).eq.0.and.ai(k).ne.0)bi=bi-ai(k)
                    lv=lv-1
                    do 12 l=k,lv
                              ai(l)=ai(l+1)
                              aj(l)=aj(l+1)
12                  continue
          endif
11    continue
c       write(u,*)(ai(k),k=1,lv),"<=",bi
c       write(u,*)(aj(k),k=1,lv),"<=",bj
c calculer la valeur du knapsack en deduire si i est redondante
      so=solve(ai,aj,bj,lv)
c       write(u,*)"so=",so
      if(so.lt.b(i))then
          test=.true.
      else
          test=.false.
      endif
      elimk=test
      return
      end
c*********************************************************************
      subroutine majr2


c met a jour ar,tar,br,cr en fonction de dtx

      implicit integer(a-z)

c definition des constantes

     parameter (nvmax=199,ncmax=101)

c definition des variables partagees

     common/donnees/nv,nc,a(ncmax,nvmax),b(ncmax),c(nvmax)
```

```
      common/e_prob/ar(ncmax,nvmax),br(ncmax),cr(nvmax)
      common/e_cont/tc(ncmax),nca,nca1,tcr(ncmax)
      common/e_var/nva,nva1,x1(nvmax),x2(nvmax)

      do 24 i=1,nc
          do 241 j=1,nv
                    ar(i,j)=a(i,j)
241       continue
24    continue

      do 22 i=1,nc
          br(i)=b(i)
22    continue

c eliminer les variables fixees en phase1
      do 9 i=1,nv
          if(x1(i).lt.2)then
                              do 23 k=i,nv
                                        do 231 l=1,nc
                                                  ar(l,k)=ar(l,k+1)
231                                     continue
23                            continue
          endif
9     continue

c eliminer les contraintes redondantes de ar
      do 10 j=1,nc
          if(tc(j).eq.0)then
                    do 21 k=j,nc
                              br(k)=br(k+1)
                              do 211 i=1,nva
                                        ar(k,i)=ar(k+1,i)
211                           continue
21                  continue
          endif
10    continue
      return
      end

      function solve(c,a,b,n)

c contient la phase 1 de la  subroutine FPK79 (Plateau) :
c calcule la partie entiere de la valeur optimale d"un probleme
c de knapsack sur [0,1].
c algorithme NKR de complexite lineaire pour n>10
c algorithme CKR pour n<=10.

      implicit integer(a-z)
      parameter(nvmax=199)
      dimension c(n),a(n)
      dimension ip(nvmax),x(nvmax)
      real val,da(nvmax)

c initialiser

      ic=0
      s=0
      iseuil=10
      k=0
      do 2 i=1,n
          ip(i)=i
2     continue
      do 10 i=1,n
          x(i)=c(i)
          da(i)=real(c(i))/real(a(i))
10    continue
```

```
      ib1=1
      ic1=n
 12   continue
      ib=ib1
      ih=ic1

      if (ih-ib.lt.iseuil)then
c algorithme CKR:
c tri des scalaires da(k)=c(k)/a(k)(tri par insertion)

          do 23 j1=ib+1,ih
                    j2=j1-1
91                  continue
                    if(j2.ge.ib)then
                              if(da(j2).lt.da(j2+1))then
                              j3=j2+1
                              ipv=ip(j2)
                              val=da(j2)
                              ival=a(j2)
                              ip(j2)=ip(j3)
                              a(j2)=a(j3)
                              da(j2)=da(j3)
                              a(j3)=ival
                              ip(j3)=ipv
                              da(j3)=val
                              j2=j2-1
                              go to 91
                              endif
                    endif
23        continue
c (da(k),k=1,n)=liste triee par ordre decroissant des valeurs de da(k)

          do 151 i=1,n
                    k13=ip(i)
                    c(i)=x(k13)
151       continue
c (c(i),i=1,n)=liste triee par ordre decroissant des valeurs de da(i)

          s=ib-1
          if(s.ne.0)then
                    do 303 i=1,s
                              ic =ic+c(i)
303                 continue
          endif
c ic contient la somme des couts deja traites par l"algorithme NKR

          i=ib
304       continue
          j=i
          k=k+a(i)
          ic=ic+c(i)
          i=i+1
          if(i.ie.ih.and.k.it.b)go to 304
c j contient l"indice du multiplicateur optimal

          ve=b-k
          ic1=j
          if(ve.ne.0)then
                    ib=j
                    r=a(ib)
                    ve=ve+a(ib)
                    ic=ic-c(ib)
          endif
c ic contient la somme des couts des variables fixees a 1
c fin de l"algorithme CKR
```

```
      else

c algorithme NKR:
c partition de J={1,...,n} en 2 ensembles U et L:
c soit  R= {da(k)=c(k)/a(k),k=1,n}
c U(J,R,k)={j,da(j)>=da(k)}
c L(J,R,k)=J-{U(J,R,k)U{k}}

          j1=ib
          j4=(ih+ib)/2
          j2=j4
          if(da(j1).gt.da(j2))then
                    j1=j4
                    j2=ib
          endif
          j3=ih
          if(da(j3).lt.da(j2))then
                    j2=ih
                    if(da(j1).ge.da(j2))j2=j1
          endif
          val=da(j2)
          ival=a(j2)
          ipv=ip(j2)
          da(j2)=da(ih)
          a(j2)=a(ih)
          ip(j2)=ip(ih)
6         continue
          if(ib.lt.ih.and.da(ib).ge.val)then
                              k=k+a(ib)
                              ib=ib+1
                              go to 6
          endif
          if(ib.lt.ih)then
                    da(ih)=da(ib)
                    a(ih)=a(ib)
                    ip(ih)=ip(ib)
4                   continue
                    ih=ih-1
                    if(ib.lt.ih.and.da(ih).le.val)go to 4
          endif
          if(ib.lt.ih)then
                              da(ib)=da(ih)
                              a(ib)=a(ih)
                              ip(ib)=ip(ih)
                              k=k+a(ih)
                              ib=ib+1
          endif
          if (ib.lt.ih) go to 6

          da(ib)=val
          a(ib)=ival
          k=k+ival
          ip(ib)=ipv

          if(k.lt.b)then
                    ib1=ib+1
                    s=k
                    go to 12
          else
                    k=k-a(ib)
          endif
          if(k.gt.b) then
                    k=s
                    ic1=ib-1
                    go to 12
          endif
```

```
c ib est l'indice du multiplicateur optimal obtenu par l'algo NKR

         do 152 i=1,n
                  k13=ip(i)
                  c(i)=x(k13)
152      continue
c (c(i),i=1,n)=liste ordonnee par da(i) par rapport a la valeur da(ib)
c si i<ib alors da(i)>=da(ib)

         ic1=ib-1
         if(ic1.ne.0)then
                  do 99 i=1,ic1
                           ic=ic+c(i)
99                continue
         endif
c ic contient la somme des couts des variables fixees a 1

         ve=b-k
         if(ve.eq.a(ib))then
                  ic1=ib
                  ic=ic+c(ib)
                  ve=0
         else
                  r=a(ib)
         endif
     endif

c la partie entiere de la  valeur en continu du knapsack
c est memorisee par solve
     solve=ic
     if(ve.ne.0)solve=(ve*c(ib)/r)+ic

     return
     end
```

4. PROBLEMES TESTS

Les expériences numériques ont été menées sur cinquante problèmes tests de la littérature. Tous possèdent les caractéristiques suivantes :

(i) l'hypothèse (H) est vérifiée

(ii) la matrice A est non négative

(iii) la matrice A est dense : $\rho > 67$ (1)

La liste de ces problèmes se décompose comme suit :

- Petersen	: 7	P	
- Hansen, Plateau	: 2	HP	(2)
- Weingartner, Ness	: 8	W	
- Shih	: 30	WS	(3)
- Senju, Toyoda	: 2	ST	
- Fleisher	: 1	F	

Notes

(1) ρ représente la densité de la matrice A ($\rho \in]0, 100]$)

$$\rho = \frac{100}{m \times n} \times \text{nombre total des } a_{ij} \text{ non nuls, } i = 1, 2, \ldots, m, \; j = 1, 2, \ldots, n$$

(2) Les 2 problèmes de Hansen et Plateau dérivent des problèmes 6 et 7 de Petersen, après utilisation d'un algorithme de réduction qui fixe à leur valeur optimale un certain nombre de variables et élimine les contraintes devenues inactives à cause de ces fixations.

(3) Tous les problèmes sont réels sauf ceux de Wei Shih. Pour ces derniers, les a_{ij} sont générés de manière aléatoire à partir d'une loi de distribution uniforme discrète sur [0,99], tandis que, les c_j sont distribués entre 1 et 999. Le choix des b_i est destiné à faire varier dans un large éventail le degré de relâchement des contraintes.

(4) Les matrices sont données ligne par ligne.

PETERSEN 1

Nombre de contraintes : 10

Nombre de variables : 6

Max {cx | Ax ≤ b ; x binaire} = 3800

c =

100	600	1200	2400	500	2000

b =

80	96	20	36	44	48	10	18	22	24

A =

8	12	13	64	22	41
8	12	13	75	22	41
3	6	4	18	6	4
5	10	8	32	6	12
5	13	8	42	6	20
5	13	8	48	6	20
0	0	0	0	8	0
3	0	4	0	8	0
3	2	4	0	8	4
3	2	4	8	8	4

x^* = 0 1 1 0 0 1

PETERSEN 2

Nombre de contraintes : 10

Nombre de variables : 10

Max {cx | Ax ≤ b ; x binaire} = 87061

c =

6001	3105	18000	38500	186	1987	8820	42000	4025	3270

b =

450	540	200	360	440	480	200	360	440	480

A =

20	5	100	200	2	4	60	150	80	40
20	7	130	280	2	8	110	210	100	40
60	3	50	100	4	2	20	40	6	12
60	8	70	200	4	6	40	70	16	20
60	13	70	250	4	10	60	90	20	24
60	13	70	280	4	10	70	105	22	28
5	2	20	100	2	5	10	60	0	0
45	14	80	180	6	10	40	100	20	0
55	14	80	200	6	10	50	140	30	40
65	14	80	220	6	10	50	180	30	50

x^* =

0	1	0	1	1	0	0	1	0	1

PETERSEN 3

Nombre de contraintes : 10

Nombre de variables : 15

Max {cx | Ax ≤ b ; x binaire} = 4015

c =

100	220	90	400	300	400	205	120	160	580
400	140	100	1300	650					

b =

550	700	130	240	280	310	110	205	260	275

A =

8	24	13	80	70	80	45	15	28	90
130	32	20	120	40					
8	44	13	100	100	90	75	25	28	120
130	32	40	160	40					
3	6	4	20	20	30	8	3	12	14
40	6	3	20	5					
5	9	6	40	30	40	16	5	18	24
60	16	11	30	25					
5	11	7	50	40	40	19	7	18	29
70	21	17	30	25					
5	11	7	55	40	40	21	9	18	29
70	21	17	35	25					
0	0	1	10	4	10	0	6	0	6
32	3	0	70	10					
3	4	5	20	14	20	6	12	10	18
42	9	12	100	20					
3	6	9	30	29	20	12	12	10	30
42	18	18	110	20					
3	8	9	35	29	20	16	15	10	30
42	20	18	120	20					

x^* =

1	1	0	1	0	1	1	0	1	1
0	0	0	1	1					

PETERSEN 4

Nombre de contraintes : 10

Nombre de variables : 20

Max {cx | Ax ≤ b ; x binaire} = 6120

c =

100	220	90	400	300	400	205	120	160	580
400	140	100	1300	650	320	480	80	60	2550

b =

550	700	130	240	280	310	110	205	260	275

A =

8	24	13	80	70	80	45	15	28	90
130	32	20	120	40	30	20	6	3	180
8	44	13	100	100	90	75	25	28	120
130	32	40	160	40	60	55	10	6	240
3	6	4	20	20	30	8	3	12	14
40	6	3	20	5	0	5	3	0	20
5	9	6	40	30	40	16	5	18	24
60	16	11	30	25	10	13	5	1	80
5	11	7	50	40	40	19	7	18	29
70	21	17	30	25	15	25	5	1	100
5	11	7	55	40	40	21	9	18	29
70	21	17	35	25	20	25	5	2	110
0	0	1	10	4	10	0	6	0	6
32	3	0	70	10	0	0	0	0	0
3	4	5	20	14	20	6	12	10	18
42	9	12	100	20	5	6	4	1	20
3	6	9	30	29	20	12	12	10	30
42	18	18	110	20	15	18	7	2	40
3	8	9	35	29	20	16	15	10	30
42	20	18	120	20	20	22	7	3	50

x^* =

1	0	0	0	0	0	0	0	0	1
0	0	0	1	1	1	1	1	1	1

PETERSEN 5

Nombre de contraintes : 10

Nombre de variables : 28

Max {cx | Ax ≤ b ; x binaire} = 12400

c =

100	220	90	400	300	400	205	120	160	580
400	140	100	1300	650	320	480	80	60	2550
3100	1100	950	450	300	220	200	520		

b =

930	1210	272	462	532	572	240	400	470	490

A =

8	24	13	80	70	80	45	15	28	90
130	32	20	120	40	30	20	6	3	180
220	50	30	50	12	5	8	18		
8	44	13	100	100	90	75	25	28	120
130	32	40	160	40	60	55	10	6	240
290	80	90	70	27	17	8	28		
3	6	4	20	20	30	8	3	12	14
40	6	3	20	5	0	5	3	0	20
30	40	10	0	5	0	0	10		
5	9	6	40	30	40	16	5	18	24
60	16	11	30	25	10	13	5	1	80
60	50	20	30	10	5	3	20		
5	11	7	50	40	40	19	7	18	29
70	21	17	30	25	15	25	5	1	100
70	55	20	50	15	15	6	20		
5	11	7	55	40	40	21	9	18	29
70	21	17	35	25	20	25	5	2	110
70	55	20	50	20	15	6	20		
0	0	1	10	4	10	0	6	0	6
32	3	0	70	10	0	0	0	0	0
30	10	0	10	10	5	0	10		
3	4	5	20	14	20	6	12	10	18
42	9	12	100	20	5	6	4	1	20
50	30	5	20	20	10	10	20		
3	6	9	30	29	20	12	12	10	30
42	18	18	110	20	15	18	7	2	40
60	50	25	25	25	15	10	28		
3	8	9	35	29	20	16	15	10	30
42	20	18	120	20	20	22	7	3	50
60	55	25	30	25	15	10	28		

x^* =

1	1	1	0	0	0	0	0	1	0
0	0	0	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	0	0

PETERSEN 6

Nombre de contraintes : 5

Nombre de variables : 39

Max {cx | Ax ≤ b ; x binaire} = 10618

c =

560	1125	300	620	2100	431	68	328	47	122
322	196	41	25	425	4260	416	115	82	22
631	132	420	86	42	103	215	81	91	26
49	420	316	72	71	49	108	116	90	

b =

600	500	500	500	600

A =

40	91	10	30	160	20	3	12	3	18
9	25	1	1	10	280	10	8	1	1
49	8	21	6	1	5	10	8	2	1
0	10	42	6	4	8	0	10	1	
16	92	41	16	150	23	4	18	6	0
12	8	2	1	0	200	20	6	2	1
70	9	22	4	1	5	10	6	4	0
4	12	8	4	3	0	10	0	6	
38	39	32	71	80	26	5	40	8	12
30	15	0	1	23	100	0	20	3	0
40	6	8	0	6	4	22	4	6	1
5	14	8	2	8	0	20	0	0	
8	71	30	60	200	18	6	30	4	8
31	6	3	0	18	60	21	4	0	2
32	15	31	2	2	7	8	2	8	0
2	8	6	7	1	0	0	20	8	
38	52	30	42	170	9	7	20	0	3
21	4	1	2	14	310	8	4	6	1
18	15	38	10	4	8	6	0	0	3
0	10	6	1	3	0	3	5	4	

x^* =

1	1	0	1	0	1	0	1	1	0
1	0	1	0	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1	0
1	1	0	1	1	1	1	1	1	

PETERSEN 7

Nombre de contraintes : 5

Nombre de variables : 50

Max {cx | Ax ≤ b ; x binaire} = 16537

c =

560	1125	300	620	2100	431	68	328	47	122
322	196	41	25	425	4260	416	115	82	22
631	132	420	86	42	103	215	81	91	26
49	420	316	72	71	49	108	116	90	738
1811	430	3060	215	58	296	620	418	47	81

b =

800	650	550	550	650

A =

40	91	10	30	160	20	3	12	3	18
9	25	1	1	10	280	10	8	1	1
49	8	21	6	1	5	10	8	2	1
0	10	42	6	4	8	0	10	1	40
86	11	120	8	3	32	28	13	2	4
16	92	41	16	150	23	4	18	6	0
12	8	2	1	0	200	20	6	2	1
70	9	22	4	1	5	10	6	4	0
4	12	8	4	3	0	10	0	6	28
93	9	30	22	0	36	45	13	2	2
38	39	32	71	80	26	5	40	8	12
30	15	0	1	23	100	0	20	3	0
40	6	8	0	6	4	22	4	6	1
5	14	8	2	8	0	20	0	0	6
12	6	80	13	6	22	14	0	1	2
8	71	30	60	200	18	6	30	4	8
31	6	3	0	18	60	21	4	0	2
32	15	31	2	2	7	8	2	8	0
2	8	6	7	1	0	0	20	8	14
20	2	40	6	1	14	20	12	0	1
38	52	30	42	170	9	7	20	0	3
21	4	1	2	14	310	8	4	6	1
18	15	38	10	4	8	6	0	0	3
0	10	6	1	3	0	3	5	4	0
30	12	16	18	3	16	22	30	4	0

x^* =

0	0	0	1	0	1	0	1	1	0
1	1	1	0	1	1	1	0	1	1
0	0	1	0	1	1	1	1	1	0
1	1	0	1	1	1	1	1	1	1
1	1	1	1	0	0	1	1	1	1

HANSEN, PLATEAU 1

Nombre de contraintes : 4

Nombre de variables : 28

Max {cx | Ax ≤ b ; x binaire} = 3418

c =

560	1125	68	328	47	122	196	41	25	115
82	22	631	132	420	86	42	103	81	26
49	316	72	71	49	108	116	90		

b =

219	203	208	180

A =

40	91	3	12	3	18	25	1	1	8
1	1	49	8	21	6	1	5	8	1
0	42	6	4	8	0	10	1		
16	92	4	18	6	0	8	2	1	6
2	1	70	9	22	4	1	5	6	0
4	8	4	3	0	10	0	6		
38	39	5	40	8	12	15	0	1	20
3	0	40	6	8	0	6	4	4	1
5	8	2	8	0	20	0	0		
38	52	7	20	0	3	4	1	2	4
6	1	18	15	38	10	4	8	0	3
0	6	1	3	0	3	5	4		

x^* =

1	1	0	1	1	0	0	1	0	1
1	1	0	0	1	0	1	0	1	0
1	0	1	1	1	1	1	1		

HANSEN, PLATEAU 2

Nombre de contraintes : 4

Nombre de variables : 35

Max {cx | Ax ≤ b ; x binaire} = 3186

c =

560	1125	620	68	328	47	122	196	41	25
115	82	22	631	132	420	86	42	103	215
81	91	26	49	316	72	71	49	108	116
90	215	58	47	81					

b =

163	165	239	168

A =

40	91	30	3	12	3	18	25	1	1
8	1	1	49	8	21	6	1	5	10
8	2	1	0	42	6	4	8	0	10
1	8	3	2	4					
16	92	16	4	18	6	0	8	2	1
6	2	1	70	9	22	4	1	5	10
6	4	0	4	8	4	3	0	10	0
6	22	0	2	2					
38	39	71	5	40	8	12	15	0	1
20	3	0	40	6	8	0	6	4	22
4	6	1	5	8	2	8	0	20	0
0	13	6	1	2					
38	52	42	7	20	0	3	4	1	2
4	6	1	18	15	38	10	4	8	6
0	0	3	0	6	1	3	0	3	5
4	18	3	4	0					

x^* =

0	0	1	0	1	1	0	1	1	0
0	1	1	0	0	1	0	1	1	1
1	1	0	1	0	1	1	1	1	1
1	1	0	1	1					

WEINGARTNER 1 à 6

Nombre de contraintes : 2

Nombre de variables : 28

Max {cx | Ax ≤ b ; x binaire} = v(B)

c =

1898	440	22507	270	14148	3100	4650	30800	615	4975
1160	4225	510	11880	479	440	490	330	110	560
24355	2865	11748	4550	750	3720	1950	10500		

A =

45	0	85	150	65	95	30	0	170	0
40	25	20	0	0	25	0	0	25	0
165	0	85	0	0	0	0	100		
30	20	125	5	80	25	35	73	12	15
15	40	5	10	10	12	10	9	0	20
60	40	50	36	49	40	19	150		

Problème	b	v(B)
1	600 600	141278
2	500 500	130883
3	300 300	95677
4	300 600	119337
5	600 300	98796
6	562 497	130623

WEINGARTNER 7 et 8

Nombre de contraintes : 2

Nombre de variables : 105

Max {cx | Ax ≤ b ; x binaire} = v(B)

c =

41850	38261	23800	21697	7074	5587	5560	5500	3450	2391
761	460	367	24785	47910	30250	107200	4235	9835	9262
15000	6399	6155	10874	37100	27040	4117	32240	1600	4500
70610	6570	15290	23840	16500	7010	16020	8000	31026	2568
2365	4350	1972	4975	29400	7471	2700	3840	22400	3575
13500	1125	11950	12753	10568	15600	20652	13150	2900	1790
4970	5770	8180	2450	7140	12470	6010	16000	11100	11093
4685	2590	11500	5820	2842	5000	3300	2800	5420	900
13300	8450	5300	750	1435	2100	7215	2605	2422	5500
8550	2700	540	2550	2450	725	445	700	1720	2675
220	300	405	150	70					

A =

75	40	365	95	25	17	125	20	22	84
75	50	15	0	0	12	0	10	0	50
0	0	10	0	0	50	60	150	0	0
75	0	102	0	0	40	60	0	165	0
0	0	45	0	0	0	25	0	150	0
0	0	158	0	85	95	0	89	20	0
0	0	0	0	0	80	0	110	0	15
0	60	5	135	0	0	25	0	300	35
100	0	0	25	0	0	225	25	0	0
0	0	0	0	0	5	0	60	0	100
0	0	0	0	0					
0	0	0	0	0	0	0	0	0	0
0	0	0	5	10	10	50	2	5	5
10	5	6	11	41	30	5	40	2	6
100	10	25	39	30	13	30	15	60	5
5	10	5	15	91	24	10	15	90	15
60	5	55	60	50	75	100	65	15	10
30	35	50	15	45	80	40	110	80	80
36	20	90	50	25	50	35	30	60	10
150	110	70	10	20	30	104	40	40	94
150	50	10	50	50	16	10	20	50	90
10	15	39	20	20					

Problème	b	v(B)
7	3000 3000	1095445
8	500 500	624319

WEI SHIH 1 à 30

Nombre de contraintes : 5

Max {cx | Ax ≤ b ; x binaire} = v(B)

c =

360	83	59	130	431	67	230	52	93	125
670	892	600	38	48	147	78	256	63	17
120	164	432	35	92	110	22	42	50	323
514	28	87	73	78	15	26	78	210	36
85	189	274	43	33	10	19	389	276	312
94	68	73	192	41	163	16	40	195	138
73	152	400	26	14	170	205	57	369	435
123	25	94	88	90	146	55	29	82	74
100	72	31	29	316	244	70	82	90	52

A_1 =

7	0	30	22	80	94	11	81	70	64
59	18	0	36	3	8	15	42	9	0
42	47	52	32	26	48	55	6	29	84
2	4	18	56	7	29	93	44	71	3
86	66	31	65	0	79	20	65	52	13
48	14	5	72	14	39	46	27	11	91
15	25	0	94	53	48	27	99	6	17
69	43	0	57	7	21	78	10	37	26
20	8	4	43	17	25	36	60	84	40

A_2 =

8	66	98	50	0	30	0	88	15	37
26	72	61	57	17	27	83	3	9	66
97	42	2	44	71	11	25	74	90	20
0	38	33	14	9	23	12	58	6	14
78	0	12	99	84	31	16	7	33	20
5	18	96	63	31	0	70	4	66	9
15	25	2	0	48	1	40	31	82	79
56	34	3	19	52	36	95	6	35	34
74	26	10	85	63	31	22	9	92	18

A_3 =

3	74	88	50	55	19	0	6	30	62
17	81	25	46	67	28	36	8	1	52
19	37	27	62	39	84	16	14	21	5
60	82	72	89	16	5	29	7	80	97
41	46	15	92	51	76	57	90	10	37
25	93	5	39	0	97	6	96	2	81
69	4	32	78	65	83	62	89	45	53
52	76	72	23	89	48	41	1	27	19
3	32	82	20	2	51	18	42	4	26

A_4 =

21	40	0	6	82	91	43	30	62	91
10	41	12	4	80	77	98	50	78	35
7	1	96	67	85	4	23	38	2	57
4	53	0	33	2	25	14	97	87	42
15	65	19	83	67	70	80	39	9	5
41	31	36	15	30	87	28	13	40	0
51	79	75	43	91	60	24	18	85	83
3	85	2	5	51	63	52	85	17	62
7	86	48	2	1	15	74	80	57	16

A_5 =

94	86	80	92	31	17	65	51	46	66
44	3	26	0	39	20	11	6	55	70
11	75	82	35	47	99	5	14	23	38
94	66	64	27	77	50	28	25	61	10
30	15	12	24	90	25	39	47	98	83
56	36	6	66	89	45	38	1	18	88
19	39	20	1	7	34	68	32	31	58
41	99	92	67	33	26	25	68	37	6
11	17	48	79	63	77	17	29	18	60

Problème	n	b					v(B)
1	30	400	500	500	600	600	4554
2	30	370	650	460	980	870	4536
3	30	480	800	500	300	620	4115
4	30	540	270	500	500	750	4561
5	30	540	240	480	600	790	4514
6	40	480	760	800	1180	940	5557
7	40	480	600	700	1200	1200	5567
8	40	480	760	800	1185	1200	5605
9	40	750	870	360	800	940	5246
10	50	850	1400	1500	450	1100	6339
11	50	880	1340	1360	300	1000	5643
12	50	850	1400	1500	440	1100	6339
13	50	850	1400	1500	400	1100	6159
14	60	1024	1700	1850	510	1310	6954
15	60	1130	420	1380	1000	1630	7486
16	60	1200	1300	630	1100	1400	7289
17	60	2090	2200	1190	2460	2320	8633
18	70	970	1310	1730	2220	2580	9580
19	70	1200	1920	2330	620	1460	7698
20	70	1320	700	1730	1954	1810	9450
21	70	1320	600	1730	1954	1810	9074
22	80	1347	2180	2683	838	1788	8947
23	80	1360	2200	2700	700	1700	8344
24	80	1100	1500	2000	2500	3000	10220
25	80	1500	800	2000	2200	2100	9939
26	90	1600	2500	3000	850	2000	9584
27	90	1600	2500	3000	900	2000	9819
28	90	1500	2500	3000	820	2000	9492
29	90	1600	2500	3000	800	2000	9410
30	90	2100	1100	3300	3700	3600	11191

SENJU, TOYODA 1 et 2

Nombre de contraintes : 30

Nombre de variables : 60

Max {cx | Ax ≤ b ; x binaire} = v(B)

c =

	2	77	6	67	930	3	6	270	33	13
	110	21	56	974	47	734	238	75	200	51
	47	63	7	6	468	72	95	82	91	83
	27	13	6	76	55	72	300	6	65	39
	63	61	52	85	29	640	558	53	47	25
	3	6	568	6	2	780	69	31	774	22

A_1 =

47	774	76	56	59	22	42	1	21	760
818	62	42	36	785	29	662	49	608	116
834	57	42	39	994	690	27	524	23	96
667	490	805	46	19	26	97	71	699	465
53	26	123	20	25	450	22	979	75	96
27	41	21	81	15	76	97	646	898	37

A_2 =

73	67	27	99	35	794	53	378	234	32
792	97	64	19	435	712	837	22	504	332
13	65	86	29	894	266	75	16	86	91
67	445	118	73	97	370	88	85	165	268
758	21	255	81	5	774	39	377	18	370
96	61	57	23	13	164	908	834	960	87

A_3 =

36	42	56	96	438	49	57	16	978	9
644	584	82	550	283	340	596	788	33	350
55	59	348	66	468	983	6	33	42	96
464	175	33	97	15	22	9	554	358	587
71	23	931	931	94	798	73	873	22	39
71	864	59	82	16	444	37	475	65	5

A_4 =

47	114	26	668	82	43	55	55	56	27
716	7	77	26	950	320	350	95	714	789
430	97	590	32	69	264	19	51	97	33
571	388	602	140	15	85	42	66	778	936
61	23	449	973	828	33	53	297	75	3
54	27	918	11	620	13	28	80	79	3

A_5 =

61	720	7	31	22	82	688	19	82	654
809	99	81	97	830	826	775	72	9	719
740	860	72	30	82	112	66	638	150	13
586	590	519	2	320	13	964	754	70	241
72	12	996	868	36	91	79	221	49	690
23	18	748	408	688	97	85	777	294	17

A_6 =

698	53	290	3	62	37	704	810	42	17
983	11	45	56	234	389	712	664	59	15
22	91	57	784	75	719	294	978	75	86
105	227	760	2	190	3	71	32	210	678
41	93	47	581	37	977	62	503	32	85
31	36	30	328	74	31	56	891	62	97

A_7 =

71	37	978	93	9	23	47	71	744	9
619	32	214	31	796	103	593	16	468	700
884	67	36	3	93	71	734	504	81	53
509	114	293	31	75	59	99	11	67	306
96	218	845	303	3	319	86	724	22	838
82	5	330	58	55	66	53	916	89	56

A_8 =

33	27	13	57	6	87	21	12	15	290
206	420	32	880	854	417	770	4	12	952
604	13	96	910	34	460	76	16	140	100
876	622	559	39	640	59	6	244	232	513
644	7	813	624	990	274	808	372	2	694
804	39	5	644	914	484	1	8	43	92

A_9 =

16	36	538	210	844	520	33	73	100	284
650	85	894	2	206	637	324	318	7	566
46	818	92	65	520	721	90	53	174	43
320	812	382	16	878	678	29	92	755	827
27	218	143	12	57	480	154	944	7	730
12	65	67	39	390	32	39	318	47	86

A_{10} =

45	51	59	21	53	43	25	7	42	27
310	45	72	53	798	304	354	79	45	44
52	76	45	26	27	968	86	16	62	85
790	208	390	36	62	83	93	16	574	150
99	7	920	860	12	404	31	560	37	32
9	62	7	43	17	77	73	368	66	82

A_{11} =

11	51	97	26	83	426	92	39	66	2
23	93	85	660	85	774	77	77	927	868
7	554	760	104	48	202	45	75	51	55
716	752	37	95	267	91	5	956	444	529
96	99	17	99	62	7	394	580	604	89
678	476	97	234	1	608	19	69	676	51

A_{12} =

410	89	414	81	130	491	6	238	79	43
5	288	910	204	948	19	644	21	295	11
6	595	904	67	51	703	430	95	408	89
11	495	844	13	417	570	9	429	16	939
430	270	49	72	65	66	338	994	167	76
47	211	87	39	1	570	85	134	967	12

A_{13} =

553	63	35	63	98	402	664	85	458	834
3	62	508	7	1	72	88	45	496	43
750	222	96	31	278	184	36	7	210	55
653	51	35	37	393	2	49	884	418	379
75	338	51	21	29	95	790	846	720	71
728	930	95	1	910	5	804	5	284	128

A_{14} =
423 6 58 36 37 321 22 26 16 27
218 530 93 55 89 71 828 75 628 67
66 622 440 91 73 790 710 59 83 968
129 632 170 67 613 608 43 71 730 910
36 92 950 138 23 95 460 62 189 73
65 943 62 554 46 318 13 540 90 53

A_{15} =
967 654 46 69 26 769 82 89 15 87
46 59 22 840 66 35 684 57 254 230
21 586 51 19 984 156 23 748 760 65
339 892 13 13 327 65 35 246 71 178
83 3 34 624 788 200 980 882 343 550
708 542 53 72 86 51 700 524 577 948

A_{16} =
132 900 72 51 91 150 22 110 154 148
99 75 21 544 110 11 52 840 201 2
6 663 22 20 89 10 93 964 924 73
501 398 3 2 279 5 288 80 91 132
620 628 57 79 2 874 36 497 846 22
350 866 57 86 83 178 968 52 399 628

A_{17} =
869 26 710 37 81 89 6 82 82 56
96 66 46 13 934 49 394 72 194 408
5 541 88 93 36 398 508 89 66 16
71 466 7 95 464 41 69 130 488 695
82 39 95 53 37 200 87 56 268 71
304 855 22 564 47 26 26 370 569 2

A_{18} =
494 2 25 61 674 638 61 59 62 690
630 86 198 24 15 650 75 25 571 338
268 958 95 898 56 585 99 83 21 600
462 940 96 464 228 93 72 734 89 287
174 62 51 73 42 838 82 515 232 91
25 47 12 56 65 734 70 48 209 71

A_{19} =
267 290 31 844 12 570 13 69 65 848
72 780 27 96 97 17 69 274 616 36
554 236 47 7 47 134 76 62 824 55
374 471 478 504 496 754 604 923 330 22
97 6 2 16 14 958 53 480 482 93
57 641 72 75 51 96 83 47 403 32

A_{20} =
624 7 96 45 97 148 91 3 69 26
22 45 42 2 75 76 96 67 688 2
2 224 83 69 41 660 81 89 93 27
214 458 66 72 384 59 76 538 15 840
65 63 77 33 92 32 35 832 970 49
13 8 77 75 51 95 56 63 578 47

A_{21} =
33 62 928 292 2 340 278 911 818 770
464 53 888 55 76 31 389 40 864 36
35 37 69 95 22 648 334 14 198 42
73 594 95 32 814 45 45 515 634 254
42 29 15 83 55 176 35 46 60 296
262 598 67 644 80 999 3 727 79 374

A_{22} =

19	780	400	588	37	86	23	583	518	42
56	1	108	83	43	720	570	81	674	25
96	218	6	69	107	534	158	56	5	938
9	938	274	76	298	9	518	571	47	175
63	93	49	94	42	26	79	50	718	926
419	810	23	363	519	339	86	751	7	86

A_{23} =

47	75	55	554	3	800	6	13	85	65
99	45	69	73	864	95	199	924	19	948
214	3	718	56	278	1	363	86	1	22
56	114	13	53	56	19	82	88	99	543
674	704	418	670	554	282	5	67	63	466
491	49	67	154	956	911	77	635	2	49

A_{24} =

53	12	79	481	218	26	624	954	13	580
130	608	3	37	91	78	743	1	950	45
41	718	36	30	534	418	452	359	759	88
29	499	55	974	93	56	108	257	93	171
13	92	63	714	9	84	890	16	930	967
748	5	7	6	327	894	33	629	448	21

A_{25} =

9	19	7	535	75	3	27	928	21	7
864	27	73	61	25	75	876	16	92	22
248	11	86	944	872	996	252	2	800	334
93	107	254	441	930	744	97	177	498	931
694	800	9	36	6	539	35	79	130	860
710	7	630	475	903	552	2	45	97	974

A_{26} =

17	36	77	843	328	22	76	368	39	71
35	850	96	93	87	56	972	96	594	864
344	76	17	17	576	629	780	640	56	65
43	196	520	86	92	31	6	593	174	569
89	718	83	8	790	285	780	62	378	313
519	2	85	845	931	731	42	365	32	33

A_{27} =

65	59	2	671	26	364	854	526	570	630
33	654	95	41	42	27	584	17	724	59
42	26	918	6	242	356	75	644	818	168
964	12	97	178	634	21	3	586	47	382
804	89	194	21	610	168	79	96	87	266
482	46	96	969	629	128	924	812	19	2

A_{28} =

468	13	9	120	73	7	92	99	93	418
224	22	7	29	57	33	949	65	92	898
200	57	12	31	296	185	272	91	77	37
734	911	27	310	59	33	87	872	73	79
920	85	59	72	888	49	12	79	538	947
462	444	828	935	518	894	13	591	22	920

A_{29} =

23	93	87	490	32	63	870	393	52	23
63	634	39	83	12	72	131	69	984	87
86	99	52	110	183	704	232	674	384	47
804	99	83	81	174	99	77	708	7	623
114	1	750	49	284	492	11	61	6	449
429	52	62	482	826	147	338	911	30	984

A_{30} =

35	55	21	264	5	35	92	128	65	27
9	52	66	51	7	47	670	83	76	7
79	37	2	46	480	608	990	53	47	19
35	518	71	59	32	87	96	240	52	310
86	73	52	31	83	544	16	15	21	774
224	7	83	680	554	310	96	844	29	61

Problème	b										v(B)
1	6000	6000	6000	6000	6000	6000	6000	6000	6000	4000	7772
	6000	6000	6000	6000	6000	6000	6000	6000	6000	4000	
	6000	6000	6000	6000	6000	6000	6000	6000	6000	4000	
2	10000	10000	10000	10000	10000	10000	10000	10000	10000	7000	8722
	10000	10000	10000	10000	10000	10000	10000	10000	10000	7000	
	10000	10000	10000	10000	10000	10000	10000	10000	10000	7000	

FLEISHER

Nombre de contraintes : 10

Nombre de variables : 20

Max {cx | Ax ≤ b ; x binaire} = 2139

c =

245	177	291	237	114	237	194	211	231	211
97	168	174	134	308	271	131	211	97	282

b =

463	451	623	493	551	647	624	511	595	526

A =

77	12	57	7	21	20	85	52	72	90
62	71	93	8	79	68	68	44	13	52
21	85	87	82	26	61	36	1	14	19
74	26	85	24	56	63	0	12	1	39
72	58	72	93	3	93	17	54	92	79
12	20	24	30	99	79	15	11	8	89
76	14	86	37	12	48	54	58	32	15
6	54	32	32	71	61	29	92	20	73
11	62	93	56	80	52	93	38	89	33
17	32	35	76	51	31	40	73	42	78
28	85	3	31	93	44	93	73	35	28
11	37	99	73	91	67	7	95	15	97
32	42	54	90	59	98	94	88	59	86
10	0	59	15	76	97	55	87	70	1
77	18	87	61	64	37	41	39	3	18
20	55	50	30	51	92	88	55	60	86
53	83	68	50	86	20	19	58	48	99
57	67	21	36	41	13	34	94	50	50
41	68	47	16	5	9	35	52	88	73
64	96	9	62	79	27	16	24	30	60

x^* = 0 1 0 1 0 1 0 1 0 1

REFERENCES

REFERENCES

[1] BALAS E. and ZEMEL E., *An algorithm for Large Zero-One Knapsack Problem,* Operations Research 28 (1980) 1130-1154.

[2] FAYARD D. and PLATEAU G., *An algorithm for the Solution of the 0-1 Knapsack Problem,* Computing 28 (1982) 269-287.

[3] FLEISCHER J., Sigmap Newsletter 20 (1976).

[4] FREVILLE A. and PLATEAU G., *Heuristics and reduction methods for multiple constraints 0-1 linear programming problems,* European Journal of Operational Research 24 (1986) 206-215.

[5] FREVILLE A. and PLATEAU G., *Hard 0-1 multiknapsack test problems for size reduction methods,* Research Report 72, Université Paris-Nord (1987).

[6] GAVISH B. and PIRKUL H., *Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality,* Mathematical Programming 31 (1985) 78-105.

[7] GLOVER F., *A multiphase dual algorithm for the 0-1 integer programming problem,* Operations Research 13 (1965) 879-919.

[8] GOPALAKRISHNAN P.S., KANAL L.N. and RAMAKRISHNAN I.V., *Approximate algorithms for the Knapsack problem on Parallel computers,* IBM report RC 12549 (1987) 1-34.

[9] HAMMER P.L., PADBERG M.W. and PELED U.N., *Constraint pairing in integer programming,* INFOR - Canadian Journal of Operational Research and Information Processing 13 (1975) 68-81.

[10] KARNIN E.D., *A parallel algorithm for the knapsack problem,* IEEE Trans. On Computers C-33 (5) (1984) 404-408.

[11] KLEIN P. and MEYER auf der HEIDE F., *A lower time bound for the knapsack problem on random acces machine,* Acta Informatica 19 (1983) 385-395.

[12] LAVALLEE I. and ROUCAIROL C., *Parallel branch and bound algorithms,* EURO VII Congress, Research Report MASI 112, Université Paris 6 (1985).

[13] MARTELLO S. and TOTH P., *Algorithm for the solution of the 0-1 single knapsack,* Computing 21 (1978) 81-86.

[14] MARTELLO S. and TOTH P., *A new algorithm for the 0-1 knapsack problem,* presented at ORSA-TIMS Meeting, Miami Beach (1986).

[15] PETERSEN C.C., *Computational experience with variants of the Balas algorithm applied to the selection of R and D projects,* Management Science 13 (9) (1967) 736-750.

[16] PLATEAU G., *Reduction de la taille des problèmes linéaires en variables 0-1,* Research Report 71, UST Lille 1 (1976).

[17] PLATEAU G. and ELKIHEL M., *A hybrid method for the 0-1 knapsack problem,* Methods of Operations Research 49 (1985) 277-293.

[18] PLATEAU G., ROUCAIROL C. and VALABREGUE I., *Algorithm PR^2 for the parallel size reduction of the 0-1 multiknapsack problem,* Research Report INRIA, Rocquencourt (to appear).

[19] ROUCAIROL C., *A parallel branch and bound algorithm for the quadratic assignment problem,* Discrete Applied Mathematics 18 (1987) 211-225.

[20] ROUCAIROL C., *Du sequentiel au parallèle : la recherche arborescente et son application à la programmation quadratique en variables 0-1,* Thèse d'Etat, Université Paris 6 (1987).

[21] SENJU S. and TOYODA Y., *An approach to linear programming with 0-1 variables,* Management Science 15 (4) (1968) 196-207.

[22] SHIH W., *A branch and bound method for the multiconstraint 0-1 knapsack problem,* Journal of the Operational Research Society 30 (4) (1979).

[23] VALABREGUE I., *Multiknapsack en parallèle : phase de réduction,* Mémoire d'Ingénieur IIE-CNAM, Evry (1987).

[24] WEINGARTNER H.M. and NESS D.N., *Methods for the solution of the multidimensional 0-1 knapsack problem,* Operations Research 15 (1) (1967) 83-103.

[25] YAO A.C., *On parallel computation for the knapsack problem,* JACM 29 (3) (1982) 898-903.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique