Improved time and space complexity for Kianfar's inequality rotation algorithm

D. El Baz*, M. Elkihel and L. Gely

LAAS-CNRS, Université de Toulouse
7, avenue du Colonel Roche
31077 Toulouse Cedex 4, France
E-mail: elbaz@laas.fr
E-mail: elkihel@laas.fr
E-mail: lgely@laas.fr
*Corresponding author

G. Plateau

LIPN, UMR 7030 CNRS
Institut Galilée
Université Paris 13
99, avenue J.B. Clément
93430 Villetaneuse, France
E-mail: gerard.plateau@lipn.univ-parisl3.fr

Abstract: In this paper, constraint rotation techniques are considered for preconditioning 0–1 knapsack problems. These techniques permit one to generate new inequalities by means of rotation of the original ones in order to approach the convex hull associated with the feasible integer points. The time and space complexities of Kianfar's inequality rotation algorithm for combinatorial problems are improved. A revisited algorithm with $\mathcal{O}(n\overline{C})$ and $\mathcal{O}(\overline{C})$, representing, time and space complexity, respectively, is proposed, where $\overline{C}$ is smaller than the knapsack capacity.

[Submitted 12 April 2008; Accepted 26 May 2008]

Keywords: knapsack problems; constraint rotation techniques; lifting; dynamic programming.

Reference to this paper should be made as follows: El Baz, D., Elkihel, M., Gely, L. and Plateau, G. (2009) 'Improved time and space complexity for Kianfar's inequality rotation algorithm', *European J. Industrial Engineering*, Vol. 3, No. 1, pp.90–98.

Biographical notes: Didier El Baz was graduated an Engineer in Electrical Engineering and Computer Science in 1981 by INSA Toulouse, France. He received the Docteur Ingénieur degree In Control Theory from INSA in 1984 and the Habilitation à Diriger des Recherches in Computer Science from the Institut National Polytechnique de Toulouse in 1998. Dr. El Baz was Visiting Scientist in the Laboratory for Information and Decision Systems, MIT, Cambridge, USA in 1984–1985. He is presently Head of the Distributed

Computing and Asynchronism team in LAAS-CNRS Toulouse, France. His domain of research includes parallel computing, optimisation, control theory and applied mathematics.

Moussa Elkihel received a Doctoral degree in Applied Mathematics from Université des Sciences et Technologies de Lille, France in 1984. Dr. Elkihel is Maître de Conférence à l'IUT de Mécanique de Toulouse. The domain of research of Dr. Elkihel includes knapsack problems and parallel computing. Dr. Elkihel was cofounder with Dr. El Baz of the working group Knapsack and Optimisation of the French Operation Research Group (GDR RO).

Laurent Gely is currently a PhD student.

Gérard Plateau received his Thèse d'Etat in Mathematical Sciences from Université des Sciences et Technologies de Lille, France in 1979. Plateau is a Professor at Institut Galilée (University of Paris 13, France) and has been a member of LIPN (Computer Science Laboratory of Paris-Nord associated with CNRS) since 1985. He leads LIPN's combinatorial optimisation team and was director of LIPN for six years and research joint manager of Institut Galilee for four years. About thirty PhD students defended their dissertations under his direction. His scientific works are in the field of integer programming, operations research and computer science. He is the author of about 50 papers in journals, book chapters or refereed proceedings and has given around a hundred talks in conferences.

1 Introduction

We concentrate on combinatorial problems of the following form:

$$\max\left\{\sum_{j=1}^{n} p_j x_j \,\middle|\, \sum_{j=1}^{n} w_{ij} x_j \le C_i, i = 1,...m; x_j \in \{0,1\}, j = 1,....,n\right\}, \tag{1}$$

where C_i denotes the capacity of the *i*-th knapsack; *m* and *n* the number of knapsacks and items, respectively; p_j the profit associated with the *j*-th item; and w_{ij}, the weight of the *j*-th item in the *i*-th knapsack (see Kellerer *et al.*, 2004; Martello and Toth, 1990; Wolsey, 1998). Without loss of generality, we assume that all data are positive integers. In order to avoid trivial solutions, we assume that we have: $\sum_{j=1}^{n} w_{ij} > C_i, i = 1,...,m,$ and $w_{ij} \le C_i$ for all $i \in \{1... m\}$ $j \in \{1, ...,n\}$.

Many real-world applications can be formulated as problem (1) (see Gavish and Pirkul, 1982; Thesen, 1973; Yang, 2001; Kacem, to appear). Problem (1) is well known to be NP complete. Some multi-knapsack problems may be very difficult to solve. In some cases, a new formulation of a problem that is intractable in reasonable time permits one to solve it more easily. Among the techniques that can be used to reformulate problems, we can quote coefficient reduction (see Bradley *et al.*, 1975), facets (see Wolsey, 1976) and rotation techniques (see Kianfar, 1971).

Constraint rotation techniques allow one to get tighter equivalent formulations of integer linear programming problems (see Kianfar, 1971; 1976). These techniques permit one to generate new inequalities by means of rotation of the original ones in order to approach the convex hull associated with the feasible integer points. These methods are generally used as a preconditioning. The basic principle of constraint rotation techniques can be presented as in the following equations. For simplicity of presentation, constraints will be denoted in the following form in the sequel:

$$wx = \sum_{j=1}^{n} w_j x_j \le C, \tag{2}$$

with $C,\ w_j \in N^*,\ j \in \{1,...,n\},\ \max_j \{w_j\} \le C < \sum_{j=1}^{n} w_j$ and $x \in S$ where

$$S = \left\{x \in \{0,1\}^n \mid wx \le C\right\}. \tag{3}$$

We now introduce the following convex polyhedra related to S:

$$\overline{S} = \left\{x \in [0,1]^n \mid \sum_{j=1}^{n} w_j x_j \le C\right\}. \tag{4}$$

The rotation of Inequality (2) consists in moving the hyperplane:

$$wx = \sum_{j=1}^{n} w_j x_j = C, \tag{5}$$

in such a way that the new hyperplane:

$$\hat{w}x = \sum_{j=1}^{n} \hat{w}_j x_j = C, \tag{6}$$

passes through more integer points in the space $\overline{S}$ than the original one, so that we obtain a stronger inequality if it is possible. More precisely, the problem consists in obtaining the largest integer $\hat{w}_i$ if it exists such that $\hat{w}_i > w_i$ and $S_i = S$, where:

$$S_i = \left\{x \in \{0,1\}^n \mid \sum_{j=1, j\ne i}^{n} w_j x_j + \hat{w}_i x_i \le C\right\}. \tag{7}$$

This process is repeated for all $i \in \{1,...,n\}$. Finally, we have:

$$S = \hat{S} = \left\{x \in \{0,1\}^n \mid \sum_{j=1}^{n} \hat{w}_j x_j \le C\right\}. \tag{8}$$

Let us now define the following sets:

$$\overline{\hat{S}} = \left\{x \in [0,1]^n \mid \sum_{j=1}^{n} \hat{w}_j x_j \le C\right\}, \tag{9}$$

$$H = \left\{x \in \{0,1\}^n \mid wx = C\right\}, \tag{10}$$

$$\hat{H} = \left\{x \in \{0,1\}^n \mid \hat{w}x = C\right\}. \tag{11}$$

The sets H and $\hat{H}$, denote the intersections of the set S with the hyperplanes $wx = C$ and $\hat{w}x = C$, respectively. Then, we have (see Kianfar, 1971; 1976):

$$\overline{\hat{S}} \subset \overline{S} \text{ and } H \subset \hat{H}. \tag{12}$$

Thus, the first relation implies that constraint rotation will permit one to obtain a stronger inequality, *i.e.*, the domain may be smaller, though it contains the same integer points. The second relation traduces the fact that the new constraint passes through as many integer points as possible.

In Section 2, we recall first the principles of the constraint rotation algorithm proposed by Kianfar (see Kianfar, 1971; 1976). Then, we propose a revisited algorithm which improves the time and space complexities. Section 3 provides a brief conclusion.

2 Efficient inequality rotation algorithm

In this section, we present an efficient algorithm which performs constraint rotation with improved time and space complexities, as compared with Kianfar's method (see Kianfar, 1971; 1976, see also Elkihel, 1984).

2.1 Constraint rotation technique

We first present the principles of the constraint rotation algorithm proposed by Kianfar. The technique used consists in performing coefficient modifications of Constraint (2) recursively as follows, starting from the n-th component, denoted by x_n, to the first one, denoted by x_1. We concentrate first on component x_n and consider the problem:

$$v(n) = \max\left\{ \sum_{j=1}^{n-1} w_j x_j \mid \sum_{j=1}^{n-1} w_j x_j \le C - w_n, x_j \in \{0,1\}, j \in \{1,...,n-1\} \right\}. \tag{13}$$

If $v(n) < C - w_n$, then constraint rotation is performed and w_n takes the new value $\hat{w}_n$ such that:

$$\hat{w}_n = w_n + (C - w_n - v(n)) = C - v(n), \tag{14}$$

so that the new hyperplane contains at least one integer point with $x_n = 1.$ Otherwise, the value of w_n remains unchanged. For $k = n-1,\ n-2,...,2,$ consider now this series of problems:

$$v(k) = \max\left\{ \begin{array}{l} \sum_{j=1}^{k-1} w_j x_j + \sum_{j=k+1}^{n} \hat{w}_j x_j \mid \sum_{j=1}^{k-1} w_j x_j + \sum_{j=k+1}^{n} \hat{w}_j x_j \le C - w_k, \\ x_j \in \{0,1\}, j \in \{1,...,k-1\} \cup \{k+1,...,n\} \end{array} \right\}. \tag{15}$$

If $v(k) < C - w_k$, then one performs rotation of constraint and w_k takes the value $\hat{w}_k$, given as follows:

$$\hat{w}_k = w_k + (C - w_k - v(k)) = C - v(k), \tag{16}$$

in such a way that the new hyperplane contains at least one integer point with $x_k = 1$. Otherwise, one does not change the value of w_k This process is repeated till $k = 2$. For $k = 1$:

$$v(1) = \max\left\{\sum_{j=2}^{n} \hat{w}_j x_j \mid \sum_{j=2}^{n} \hat{w}_j x_j \le C - w_1, x_j \in \{0,1\}, j \in \{2,...,n\}\right\}. \tag{17}$$

If $v(1) < C - w_1$, then constraint rotation is performed and w_1 takes the new value $\hat{w}_1$. such that:

$$\hat{w}_1 = w_1 + \left(C - w_1 - v(1)\right) = C - v(1), \tag{18}$$

so that the hyperplane will contain at least one integer point with $x_1 = 1$. Otherwise, in a way similar to the situations quoted above, the value of w_1 remains unchanged.

We recall that the Kianfar algorithm, which is based on the above technique, has $\mathcal{O}(n^2C)$ time complexity and $\mathcal{O}(nC)$ space complexity (see Kianfar, 1971; 1976).

In the next subsection, we propose a revisited constraint rotation algorithm based on dynamic programming with improved time and space complexities.

2.2 Revisited constraint rotation algorithm

The algorithm relies on the technique presented in the previous subsection. Components will be computed using the dynamic programming list method. The algorithm is decomposed into two steps. In the first step, Problem (13) is solved using the dynamic programming list method (see Ahrens and Finke, 1975; Plateau and Elkihel, 1985). The list generated by the first step is used in order to prepare computations performed during the second step, where series of Problems (15), $k = n-1, n-2, ..., 2$ and (17), for $k = 1$, are solved using the dynamic programming list method.

Step 1

The dynamic programming list method proposed by Ahrens and Finke (see Ahrens and Finke, 1975) is based on the concepts of list and dominance. In Step 1, we shall recursively generate lists L_k of pairs (w, p), $k = 1, 2, ... , n-1$, where w is a weight and $p \le k$ is the dynamic programming stage number at which pair (w, p) was created. Initially, we have $L_0 = \{(0,0)\}$.

Let us define first the set N_k of new pairs generated at stage k, where new pairs result from the fact that a new item, *i.e.*, the k-th item, was taken into account. We have:

$$N_k = \left\{\left(w + w_k, k\right) \mid \left(w, p\right) \in L_{k-1}, w + w_k \le \overline{C}\right\}, \tag{19}$$

where

$$\overline{C} = C - \min_{j \in \{1,...,n\}} w_j. \tag{20}$$

Note that $\overline{C}$ is introduced here in order to diminish the total work. According to the dominance principle, which is a consequence of Bellman's optimality principle, all pairs $(w, p) \in L_{k-1} \cup N_k$ obtained by construction such that there exists a pair

$(w', p') \in L_{k-1} \cup N_k$, which satisfies: $w = w'$ and $p' < p$, must not belong to a list L_k. In this case, we usually say that the pair (w', p') dominates the pair (w, p). Thus, we can define as follows the set D_k of dominated pairs at stage *k*:

$$D_k = \{(w, p) \mid (w, p) \in L_{k-1} \cup N_k, \exists (w', p') \in L_{k-1} \cup N_k \text{ with } w = w', p' < p\} \tag{21}$$

For all positive integers k, the dynamic programming recursive list L_k is defined as follows:

$$L_k = L_{k-1} \cup N_k - D_k. \tag{22}$$

Note that the lists L_k are organised as sets of monotonically increasing ordered pairs by weight. From the list L_{n-1} we can derive the value of w_n according to Equation (14) with:

$$v(n) = \max\{w \mid w \leq C - w_n, (w, p) \in L_{n-1}\}. \tag{23}$$

Remark 1. Note that only one list is stored in the memory, *i.e*., the currently growing list L_k. Thus, at the end of Step 1, all data necessary to perform Step 2 will be contained in the list L_{n-1}. Note also that the lists L_k of pairs *(w,p)* can be derived easily from L_{n-1} as follows, using the index *p*:

$$L_k = \{(w, p) \in L_{n-1} \mid p \leq k\}. \tag{24}$$

As a consequence, the space complexity of Step 1 is $\mathcal{O}(\overline{C})$. Note that the time complexity of computing *v(n)* is $\mathcal{O}(\overline{C})$. Note also that each list L_k is generated with time complexity $\mathcal{O}(\overline{C})$. Thus, the time complexity of Step 1 is $\mathcal{O}(n\overline{C})$.

Step 2

In a way quite similar to Step 1, we can recursively generate lists $\hat{L}_k$ of weights *w*, starting from $k = n$ to $k = 2$. Let us consider first the set $\hat{L}_{n+1}$ such that:

$$\hat{L}_{n+1} = \{0\}. \tag{25}$$

We define as follows the set $\hat{N}_k$ of new weights generated at stage *k*, where new weights result from the fact that a new item, *i.e*., the *k*-th item, is taken into account at this stage:

$$\hat{N}_k = \{\hat{w} + \hat{w}_k \mid \hat{w} \in \hat{L}_{k+1}, \hat{w} + \hat{w}_k \leq \overline{C}\}. \tag{26}$$

According to the dominance principle, all $\hat{w} \in \hat{L}_{k+1} \cup \hat{N}_k$ obtained by construction, such that there exists a $\hat{w}' \in \hat{L}_{k+1} \cup \hat{N}_k$, which satisfies: $\hat{w} = \hat{w}'$, must not belong to a list $\hat{L}_k$. In this case, we usually say that $\hat{w}'$ dominates the weight $\hat{w}$. Thus, we can define the set $\hat{D}_k$ of dominated weights at stage *k* as follows:

$$\hat{D}_k = \left\{ \hat{w} \mid \hat{w} \in \hat{L}_{k+1} \cup \hat{N}_k, \exists \hat{w}' \in \hat{L}_{k+1} \cup \hat{N}_k \text{ with } \hat{w} = \hat{w}' \right\}.$$

As a consequence, for all positive integers k, the dynamic programming recursive list $\hat{L}_k$ is denned as follows:

$$\hat{L}_k = \hat{L}_{k+1} \cup \hat{N}_k - \hat{D}_k. \tag{27}$$

Note that the lists $\hat{L}_k$ are organised as sets of monotonically increasing weights.

From the lists L_{n-1} and $\hat{L}_k$, we can derive the value $\hat{w}_{k-1}$, $k = n, n-1, \ldots, 3,$ according to Equation (16) with:

$$v(k-1) = \max \left\{ \begin{array}{l} w + \hat{w} | w + \hat{w} \le C - w_{k-1}, \hat{w} \in \hat{L}_k \\ \text{and } (w, p) \in L_{n-1}, \text{ with } p \le k-2 \end{array} \right\}. \tag{28}$$

From the list $\hat{L}_2$, we can compute the value of $\hat{w}_1$ according to Equation (18) with:

$$v(1) = \max \left\{ \hat{w} | \hat{w} \le C - w_1, \hat{w} \in \hat{L}_2 \right\}. \tag{29}$$

Remark 2. Similarly to Step 1, only one list is stored in the memory during Step 2, *i.e.*, the current list $\hat{L}_k$. As a consequence, the Step 2 space complexity is $\mathcal{O}(\bar{C})$. The computation of $v(k-1)$ is performed by merging the lists L_{n-1} and $\hat{L}_k$ with time comlexity $\mathcal{O}(\bar{C})$; *i.e.*, the ordered lists L_{n-1} and $\hat{L}_k$ are simultaneously examined once. The former list is examined in an increasing way; the latter in a decreasing way (see Ahrens and Finke (1975)). Note also that each list L_k is generated with time complexity $\mathcal{O}(\bar{C})$. Thus, the Step 2 time complexity is $\mathcal{O}(n\bar{C})$. As a result, the time complexity of the global method is $\mathcal{O}(n\bar{C})$ and the space complexity is $\mathcal{O}(\bar{C})$. We recall that the time and space complexities of Kianfar's method are $\mathcal{O}(n^2C)$ and $\mathcal{O}(nC)$ respectively (see Kianfar, 1971; Kianfar, 1976).

In order to illustrate our algorithm, we now display a simple example. Let us consider the following constraint:

$$5x_1 + 11x_2 + 3x_3 + 6x_4 \le 12. \tag{30}$$

We note that the associated hyperplane passes through no point with integer components.

We have: $\bar{C} = 12 - 3 = 9$.

Step 1

$$L_0 = \{(0,0)\}.$$

$$L_1 = \{(0,0), (5,1)\}.\ L_2 = \{(0,0), (5,1)\}.$$

$$L_3 = \{(0,0), (3,3), (5,1)(8,3)\}.$$

$$C - w_4 = 12 - 6 = 6.$$

Using Equation (23), we have: $v(4) = \max\{w \mid w < 6, (w, p) \in L_3\}$. Thus, $v(4) = 5$.

$$\hat{w}_4 = C - v(4) = 12 - 5 = 7.$$

Step 2

$$\hat{L}_5 = \{0\}.$$

$$\hat{L}_4 = \{0, 7\}.$$

$$C - w_3 = 12 - 3 = 9.$$

Using Equation (28), we have: $v(3) = \max\{w + \hat{w} \mid w + \hat{w} \le 9,\ \hat{w} \in \hat{L}_4 \text{ and } (w, p) \in L_3,$ with $p < 2\}$. Thus, $v(3) = 7$.

$$\hat{w}_3 = C - v(3) = 12 - 7 = 5.$$

$$\hat{L}_3 = \{0, 5, 7\}.$$

$$C - w_2 = 12 - 11 = 1.$$

We can compute $v(2)$ as shown above. We obtain: $v(2) = 0$.

$$\hat{w}_2 = C - v(2) = 12 - 0 = 12.$$

$$\hat{L}_2 = \{0, 5, 7\}.$$

$$C - w_1 = 12 - 5 = 7.$$

Using Equation (29), we have: $v(1) = 7$.

$$\hat{w}_1 = C - v(1) = 12 - 7 = 5.$$

Thus, after constraint rotation, we obtain the following new constraint:

$$5x_1 + 12x_2 + 5x_3 + 7x_4 \le 12; \tag{31}$$

note that the associated new hyperplane now passes through the following three points with integer components: (1, 0, 0, 1), (0, 1, 0, 0), (0, 0, 1, 1).

3 Conclusion

In this paper, we have proposed a significant improvement for constraint rotation algorithms. The time and space complexities of the revisited algorithm we propose are $\mathcal{O}(n\overline{C})$ and $\mathcal{O}(\overline{C})$, respectively, where n denotes the number of variables and $\overline{C}$ is smaller than the capacity of the knapsack.

Finally, we also note that the constraint rotation algorithm proposed in this paper lends itself very well to parallel computing. As a matter of fact, the different constraints can be treated independently via several processors. Moreover, the dynamic programming lists method used in Steps 1 and 2 are also well suited to parallel computing (see El Baz and Elkihel (2005)); this permits one to obtain a $\mathcal{O}\left(\frac{nC}{q}\right)$ time complexity, where q denotes the number of processors.

References

Ahrens, J.H. and Finke, G. (1975) 'Merging and sorting applied to the zero-one knapsack problem', *Operations Research*, Vol. 23, pp.1099–1109.

Bradley, J.G.H., Hammer, P.L., Wolsey, L.A. and Finke, G. (1975) 'Coefficient reduction for inequalities in 0–1 variables', *Mathematical Programming 7*, pp.263–282.

El Baz, D. and Elkihel, M. (2005) 'Load balancing methods and parallel dynamic programming algorithm using dominance technique applied to the 0–1 knapsack problem', *Journal of Parallel and Distributed Computing*, Vol. 65, pp.74–84.

Elkihel, M. (1984) 'Programmation dynamique et rotation de contraintes pour les problèmes d'optimisation entiere', *Thèse de l'Université des Sciences et Techniques de Lille*.

Gavish, B. and Pirkul, H. (1982) 'Allocation of data bases and processors in a distributed computing system', in J. Akoka (Ed.) *Management of Distributed Data Processing*, North-Holland, pp.215–231.

Kacem, I. (to appear) 'Approximation algorithms for the makespan minimization with positive tails on a single machine with fixed non-availability interval', *Journal of Combinatorial Optimization*.

Kellerer, H., Pferschy, U. and Pisinger, D. (2004) *Knapsack Problems*, Springer.

Kianfar, F. (1971) 'Stronger inequalities for 0,1 integer programming using knapsack functions', *Operations Research*, Vol. 19, pp.1374–1392.

Kianfar, F. (1976) 'Stronger inequalities for 0–1 integer programming: computational refinements', *ORSA*, Vol. 24, pp.581–585.

Martello, S. and Toth, P. (1990) *Knapsack Problems*, New York: John Wiley & Sons.

Plateau, G. and Elkihel, M. (1985) 'A hybrid method for the 0–1 knapsack problem', *Methods of Operations Research*, Vol. 49, pp.277–293.

Thesen, A. (1973) 'Scheduling of computer programs in a multiprogramming environment', *BIT*, Vol. 13, pp.208–216.

Wolsey, L.A. (1976) 'Facets and strong valid inequalities for integer programs', *Operations Research*, Vol. 24, pp.367–372.

Wolsey, L.A. (1998) *Integer Programming*, New York: John Wiley & Sons.

Yang, M.H. (2001) 'An efficient algorithm to allocate shelf space', *European Journal of Operation Research*, Vol. 131, pp.107–118.