A Heuristic for the Multi-knapsack Problem

JOSE GRANDÓN[1] AND IVAN DERPICH[2]
Industrial Engineering Department,
University of Santiago of Chile
Ave. Ecuador 3769, Estacion central, Santiago
CHILE
ivan.derpich@usach.cl http://www.industria.usach.cl/

Abstract: - In this work a heuristic for the problem multi-knapsack , based on directions of ascent is presented. These directions are generated from a center of the polyhedron and they conduct to good approximations of the integer solutions. For it a center of the polyhedron of the relaxed problem is obtained. Then an interior ellipse is constructed in this polyhedron and those eigenvectors of the ellipse that present the best objective ascent of the function are selected as ascent direction. For determine how many eigenvectors to use, an angle that relate the eigenvector with the objective function, was used. The heuristic algorithm has been proved with problems from the OR-library. Four groups of problems were proved with 30 instances every one, combining 100 and 250 variables with 5 and 10 constraints. The results show process time that are from a little seconds for little problems, to 400 seconds for bigger problems. The Cpu time average is 190 seconds. The errors of the best solution found measured using the integrality gap are in order to 3% in the worst case.

Key-Words: - Heuristics, Integer Programming, Multi-knapsack

1 Introduction

The NP-hard multidimensional knapsack problem (MKP) presented in [2] arises in several practical problems, such as capital budgeting, cargo loading, stock cutting and processor allocation. It can be defined as

(MKP)

$$P(A,b,c)=\left\{Max \quad c^T x : Ax \le b; c>0; a_{ij}\ge 0; b_j>0; x_j\in\{0,1\}; j=1,\dots,n\right\}$$

We will use a related program called the Relaxed program, which is obtained by replacing the binary variables with others between zero and one. Let $P_R(A,b,c)$ be the relaxed problem; then,

$$P_R(A,b,c)=\left\{c^T x : Ax\le b; c>0; a_{ij}\ge 0; b_j>0; 0\le x_j\le 1; j=1,\dots,n; x_j\in\Re\right\}$$

We propose a heuristic algorithm based on interior searching in the style of well-known interior point methods, like the projective method or the ellipsoid method. The main idea of the heuristic algorithm is to locate an interior point first and then find a “good” search direction, as in the Augmentation Algorithm proposed by Spille and Weismantel [3]. In addition, we propose using a set of vectors as search directions, which have a reasonable probability of finding the vertex of the unit hypercube defined by the binary values of the variable. While moving in augmentation directions will improve the quality of the solution because the goal is to find the best vertex of the unit hypercube in terms of the objective function. For this purpose, we fix an ellipse at the center point of the interior of the polyhedron of the relaxed problem $P_R(A,b,c)$. It is known that an ellipse's Dikin has a similar form to the polyhedron defined by the matrix of the ellipse. This characteristic shows that there is a probability that the semi-axes of the ellipse pass through the vertex of the unit hypercube. If so, then we find an integer point, but if not, then we have a non-integer point over a facet of the unit hypercube. We found that this kind of points is to a distance bounded by a function of type $f(n)$, thing that "may be" nearby to an integer point. Therefore, we use a simple rounding procedure, defining a limit value over which the fractional variable is rounded to one, and the feasibility is also checked. In the proposed heuristic algorithm, we use the eigenvectors of the matrix of the ellipse instead of the semi-axes, because they require less CPU time to calculate. The number of eigenvectors to be used is defined in terms of the size and complexity of the problem. This results in the following dilemma: using every eigenvector is sure to yield the best solution, but it will be very costly in CPU time. On the other hand, using only the best eigenvector in terms of the objective function requires very little CPU time, but will yield only one solution. Therefore, we adopt a solution for this dilemma that involves balancing the CPU time and the quality of the solution. We construct an empirical rule that takes into account the size and complexity of the problem.

At this stage, we have a "good" integer point, but we are not sure if it is the optimal point. In many cases, we find that points of this kind do not use all available resources. For this reason, we implement a second stage of the algorithm to improve the solution using the slack in the constraints. The point obtained in the second stage definitely cannot be improved by only changing one component. Nevertheless, improved solutions can be obtained by interchanging components. Those components with a value of one must be set to zero in order to free up some resources, and then other components with a value of zero must be selected and set to the value of one. This computational procedure is relatively simple and does not require too much CPU time.

This paper is organized as follows: the first section introduces the main aspects of the proposed heuristic, and the second explains how an interior ellipse representing the shape of the polyhedron and the eigenvector are good candidates to be used as augmentation directions. The heuristic algorithm is presented in the third section, including the first construction step and the subsequent improvement steps. Then, in the fourth section, we present the computational results showing that the average integrality gap is on the order of 1.98%. Finally, the fifth section presents some concluding remarks.

2 Problem Formulation

Our methodology starts by fixing a central point and then moving across the chosen search directions. An augmentation direction is a direction such moving along it always increases the objective function. Based on the augmentation algorithm, we know that it is possible to find such directions in the interior of a polyhedron [3]. We found that some of the eigenvectors of the interiors of ellipses were of this type, and we constructed a procedure to find these directions. Next, we briefly review some background material.

2.1 Eigenvectors of interior ellipses

Let us define a Dikin's ellipse in the interior of the polyhedron defined as $P_R(A,b,c)$, to be

$$E = \left\{x \in \Re^n : \left(x - x^0\right)^T Q_E \left(x - x^0\right) \leq 1\right\}. \qquad (1)$$

The matrix of the ellipse *E*, Q_E is

$$Q_E = \nabla^2 \rho(x^0) = A^T D(x^0)^{-2} A \qquad (2)$$

$$\text{with } D(x^0) = diag(b_i - \alpha_i^T x^0) \ldots \qquad (3)$$

The center x^0 may be defined in several ways. The best definition for our objective is to use the analytic center, which is defined based on a logarithmic barrier function. Thus, x^0 is the point that minimize

$$\rho(x) = -\sum_{i=1}^{m} \log\left(b_i - \alpha_i^T x\right), \qquad (4)$$

where α_i is the i^{th} row of the matrix A and b_i is the i^{th} component of vector *b*. However, calculating x^0 can take a lot of CPU time. For this reason, the pseudo-Chebyshev center will be used because it can be calculated rapidly [6]. This centre only requires a simple heuristic based on the resolution of a linear program. The pseudo-Chebyshev center is the center of the maximum volume sphere inscribed in the polyhedron. This center is obtained by solving the following linear problem:

$$\max\left\{t : \sum a_{ij} x_j + t \leq b_i, \forall i; 0 \leq x_i \leq 1, \forall j\right\} \ldots .(5)$$

The ellipse Q_E is contained in the polyhedron defined by the convex hull of the extreme points of $P_R(A,b,c)$. Let $\mu_1,\ldots,\mu_n$ be the eigenvectors of the matrix Q_E. Then, we select those that are the best augmentation directions with respect to the objective function. To make this decision, we build a ranking of augmentation directions based on the following measure:

$$\beta_i = \frac{c^T \mu_i}{\|c\|_2}, \qquad ..(6)$$

where β_i is a measure of the amount of augmentation of the direction defined by μ_i. A direction may be very good from the point of view of augmentation, but can fail to lead to a integer feasible point. For this reason, it is necessary to evaluate several augmentation directions. How many directions are necessary to evaluate to find the optimum solution remains an open question; however, we know that the answer is related to the size and complexity ratio τ of the problem. The ratio τ is the maximum ratio between the right size and the sum of the coefficients of the constraints. We solve this question empirically by constructing a rule based on computational experiments. First, we define the complexity ratio τ as

$$\tau = \min_i \left\{\tau_i : \tau_i = b_i \Big/ \sum_{j=1}^{n} a_{ij}\right\} \qquad (7)$$

Then, *I* can be defined as the number of augmentation search directions.

$$I = \frac{n^{5/4}}{\tau(n-m)} \cdots \qquad (8)$$

This number I is the number of directions of the ranking that we will use to find a “good” approximation point. A “good” approximation point is a point near a vertex of the unit hypercube. We can be sure that a point obtained using an augmentation direction intersects a facet of the unit hypercube at a point near some vertex of the same unit hypercube. The distance between any two points in a facet of the unit hypercube is bounded by a certain expression in terms of the dimension of the problem, as follows.

Proposition 1: Let y be a point of a facet of the unit hypercube. The Euclidean distance from point y to the nearest vertex of the unit hypercube is bounded by $\frac{\sqrt{n-1}}{2}$. (9)

Proof: Let z be a vertex of the unitary hypercube. The components of this vector are 0 or 1. Let y be the most distant vector from the set of all possible vertices. This vector is $y=(0.5,\dots,y_i,\dots,0.5)$. Note that the point y has a component that is 0 or 1 depending on whether it lies in the facet defined by $y_i=0$ or $y_i=1$. Let $d(z,y)$ denote the Euclidean distance between the two vectors z and y. Then,

$$d(z,y)=\sqrt{(z_1-0.5)^2+\dots+(z_i-y_i)^2+(z_n-0.5)^2}$$

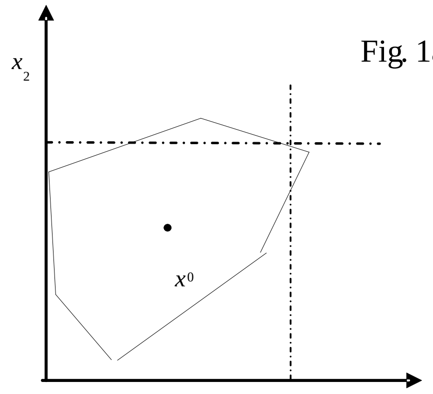
The difference between any component z_j of the vector z and the corresponding component y_j, $\forall j\neq i, j=1,\dots,n$, is 0.5, because z_j can only be equal to 0 or 1. Moreover, $z_i=y_i$, because both are in the same facet. Thus,

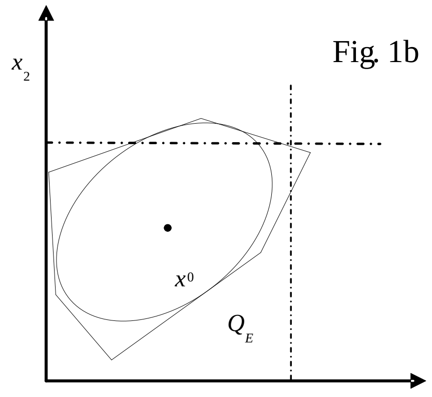
$$d(z,y)=\sqrt{(z_1-0.5)^2+\dots+(z_i-y_i)^2+(z_n-0.5)^2}=\sqrt{(n-1)(0.5)^2}=\frac{\sqrt{n-1}}{2}$$

If the coordinate vectors are used as augmentation vectors starting from the center of the unitary hypercube, the point obtained is of the form of the point y in Proposition 1, and the distance to the nearest vertex is $\frac{\sqrt{n-1}}{2}$.

However, if we use vectors corresponding to the semi-axes of an ellipse's Dikin, then there is a higher probability of finding a vertex of the unitary hypercube or of finding a non-integer point close to an integer point.

Figure 1 shows the heuristic algorithm in six graphics. The first picture shows the center point x^0 (Fig. 1a), and the second includes the ellipse Q_E (Fig. 1b).





Then, we show the eigenvector μ_1 y μ_2 of the matrix of the ellipse Q_E (Fig. 1c). The fourth picture illustrates how the eigenvectors are used as search directions (Fig. 1d). The fifth graphic shows the points found with the eigenvectors

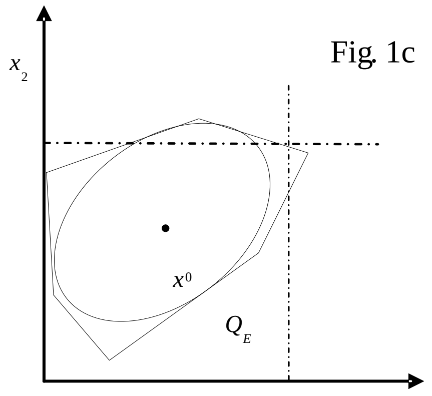


Fig. 1c

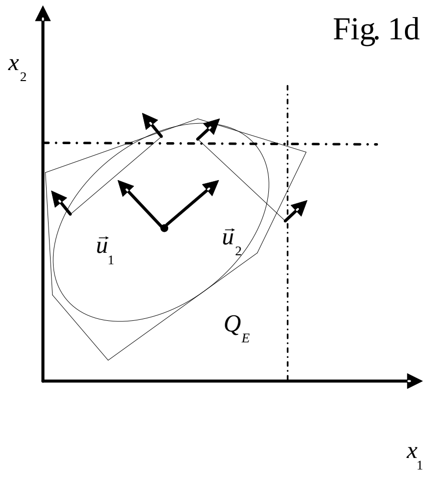


Fig. 1d

(Fig. 1e), and finally the approximation points are shown (Fig. 1f).

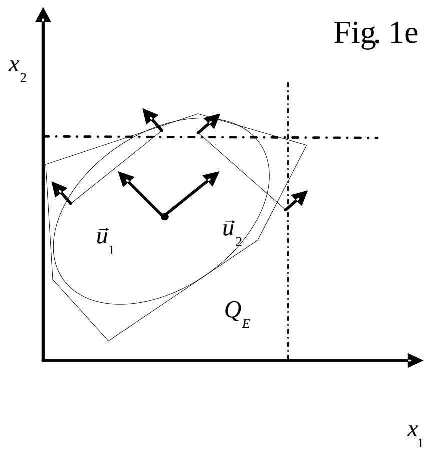


Fig. 1e

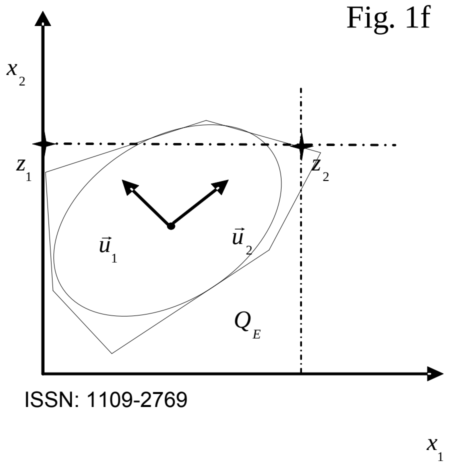


Fig. 1f

2.2 Heuristic Algorithm

The heuristic algorithm has three steps. The first is the most important, to find a point near the final integer point. The second step is to improve the solution obtained in the first step by using the right amount of slack. In the third step, a simple permutation is executed to improve the solution. The goal in steps 2 and 3 is to remove naive solutions that the first stage could generate. The heuristic algorithm uses two complementary subroutines, first to find a center point and then to round the solution to an integer point.

SUBROUTINE: *Find_Center*

INPUT: $A \in \Re^{mxn}, b \in \Re^m$

<u>OUTPUT</u>: the center point x^0

Solve the linear program:

$$\max\ t$$

$$Ax^0 + te \le b,$$
$$t \ge 0$$

where e is the vector $(1,...,1)^T$. x^0 is the optimal solution of this program.

SUBROUTINE: *Round*

INPUT: $y_k \in [0,1]^n$

<u>OUTPUT</u>: $z_k \in \{0,1\}^n$

```
For every $j = 1,...,n$
        If $y_k(j) \ge 0.9$
        Then $z_k(j) = 1$
        Else $z_k(j) = 0$
End for
```

MAIN ROUTINE: HEURISTIC ALGORITHM

```
INPUT : matrix $A \in \Re^{mxn}$, vector $b \in \Re^{m}$, vector $c \in \Re^{n}$
OUTPUT: $y$ (integer solution)
Step 1, Output: $z^1 \in Z^n$
1. Call to Subroutine Find_Center (A,b) and return $x^0$.
2. Let $Q(x^0) = \nabla^2\varphi(x^0) = A^T D(x^0)^{-2} A$, where $D(x^0) = diag(b_i - \alpha_i^T x^0)$.
3. Let $\mu_1,...,\mu_n$ be the eigenvectors of the matrix $Q(x^0)$ that will be used as search directions.
4. Let $\beta_1,...,\beta_n$ be measures of improvement along the search directions $\mu_1,...,\mu_n$, calculated as $\beta_i = \left|\frac{c^T * \mu_i}{\|c\|_2}\right|; \forall i$.
5. Let $\tau = \min\left\{\tau_i : \tau_i = b_i \Big/ \sum_{j=1}^{n} a_{ij}\right\}$.
6. Let $I = \frac{n^{5/4}}{\tau(n-m)}$.
7. Let $S$ be a vector formed by $\mu_i$ corresponding to the $I$ major measures of $\beta_i$.
8. Let $fo_{\max} = 0$.
9. For every $\mu_k \in S$,
    Solve two linear programs:
        Max/Min $\mu_k^T y_k$
        $A * y_k \le b$
        $y_k \ge 0$, with $y_k \in \Re^n$
    Call to Subroutine Round( $y_k$ ) and return $z_k \in \{0,1\}^n$.
    If ( $A * z_k \le b$ ) and $c^T * z_k > fo_{\max}$, then
        $fo_{\max} = c^T * z_k$ ; $z^1 = z_k$
End for
Step 2, Input: $z^1 \in \{0,1\}^n$, Output: $z^2 \in \{0,1\}^n$
10. Let $h = b - A * z^1$.
11. Let $J = \{ j : z^1(j) = 0 \}$ and let $I$ be a set of indices in J, ordered from the largest to the smallest values of the objective function.
12. Let $z^2 = z^1$.
13. For every $i \in I$
        $z^2(i) = 1$
        If $h \le 0$
        Then $z^2(i) = 0$
        Else
    End For
Step 3, Input: $z^2 \in \{0,1\}^n$, Output: $z^3 \in \{0,1\}^n$
14. Let $z^3 = z^2$.
15. For every $j \in J$
        If $z^3(j) = 0$
        Then $j \to J_0$
        Else $j \to J_1$
    End For
```

```
16. Order the set $J_0$ from largest to smallest values of the objective function.
17. For every $j_0 \in J_0$
        For every $j_1 \in J_1$
            If $c(j_0) > c(j_1)$
            Then
                If $h + A_{j_1} - A_{j_0} \geq 0$
                Then
$j_1 \to J_0$ ; $j_0 \to J_1$
        End For
    End For
18. For Every $j = 1,...,n$
        If $j \in J_1$
        Then $z^3(j) = 1$
        Else $z^3(j) = 0$
    End For
```

Where A_j is the j-jth column of matrix A.

A simple complexity analysis shows that the main routine makes $2n$ calls to the simplex algorithm in the first step, and in the second step it performs n matrix operations. In the third step, $2n$ matrix operations are required.

The subroutine round performs n operations, and the subroutine *find_center* solves a simplex problem. Thus, we can say that the complexity is on the order of $O(2n)$ calls to the simplex problem.

3. Computational Experiments

The heuristic algorithm was tested using test problems taken from the public web OR-library [1] for instances of the MKP. Nine groups of 30 instances were tested, generating a total of

Table 1. Results grouped by variables, constraints and tightness.

n	m	Tighness ratio (%)	Number of iterations	CPU Time (s)	Gap (%)
100	5	25%	13	5,9	3,40%
100	5	50%	7	5,1	1,33%
100	5	75%	4	2,3	0,38%
100	10	25%	14	7,3	6,90%
100	10	50%	7	5,3	1,05%
100	10	75%	5	4,8	0,65%
100	30	25%	18	9,3	4,95%
100	30	50%	9	7,4	3,62%
100	30	75%	6	6,4	1,01%
250	5	25%	16	55,2	5,02%
250	5	50%	8	27,9	0,72%
250	5	75%	5	18,1	0,44%
250	10	25%	17	63.1	5,69%
250	10	50%	8	31,5	0,98%
250	10	75%	6	24,5	0,50%
250	30	25%	18	66	7,04%
250	30	50%	9	38,6	2,51%
250	30	75%	6	29,2	0,81%
500	5	25%	19	546,6	0,76%
500	5	50%	10	295,5	0,12%
500	5	75%	6	186,9	0,29%
500	10	25%	19	558,8	1,01%
500	10	50%	10	291	0,26%
500	10	75%	6	182,8	0,18%
500	30	25%	20	609,4	2,75%
500	30	50%	10	314,1	0,70%
500	30	75%	6	192,2	0,32%
Average			13	135,5	3,40%

270 instances. These groups were formed by combining three possible numbers of variables (100, 250, 500), three possible numbers of constraints (5, 10, 30) and three possible values for the ratio of tightness Γ (25%, 50%, 75%).

The algorithm was implemented using a personal computer with a processor of 1.8 GHz and 1 GB RAM. The instances were solved

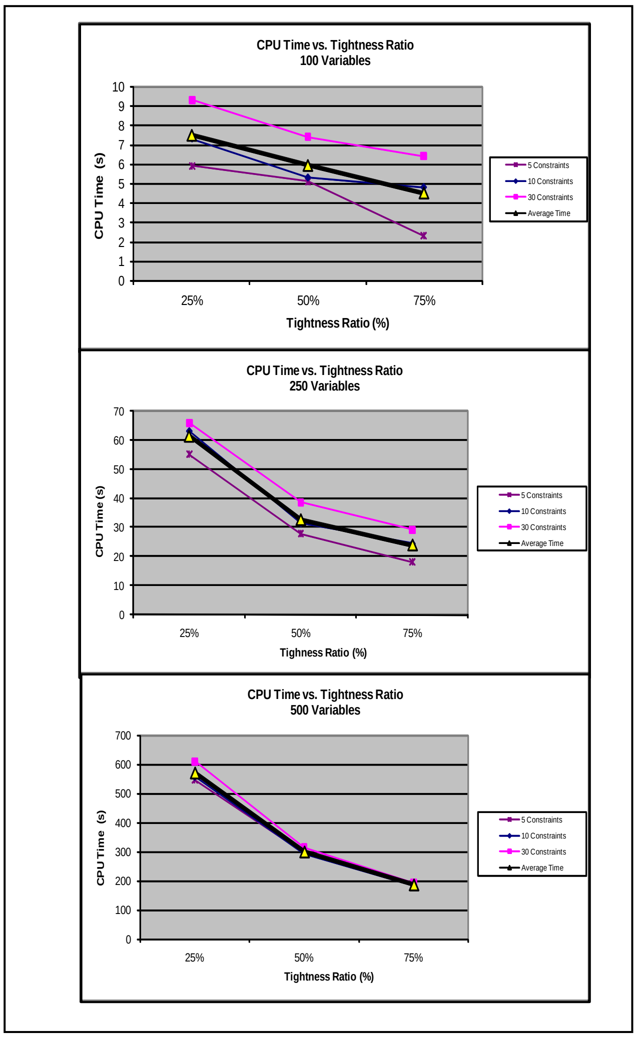
using Matlab software version R12. The results are shown in Table 1 in terms of the number of variables, restrictions, and tightness ratios.
The results shown in Table 1 show an average integrality gap of 1.98%. This means that the percent difference between the values obtained for the heuristic algorithm and the optimal solution is lower than 2%. This is a good value considering the values of the function objective. In particular, the average gap is less than the maximum coefficient of the objective function. Moreover, the CPU times are reasonable in comparison with the CPU times required for an optimal solution. The average CPU time is 132.71 seconds, or equivalently, 2.2 minutes.

3.1. CPU Time

Graph 1 below shows the CPU times of the heuristic algorithm in seconds for the different numbers of variables, the ratio of the problems and the number of restrictions. Observe that the CPU times are inversely related to the complexity of the problem measured by τ , so in these cases, more CPU time is required for those problems that are geometrically more difficult in terms of tightness.

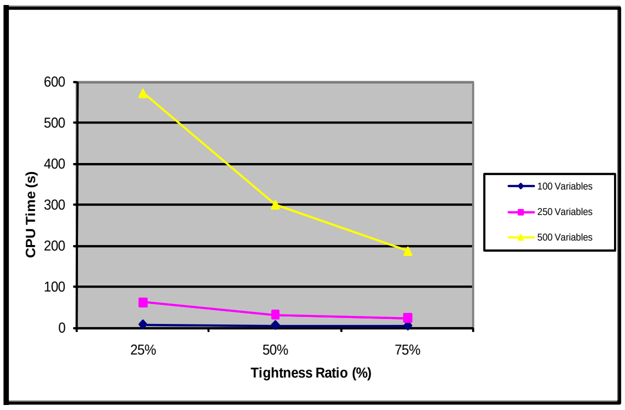
This finding may be caused by the rule used in the heuristic algorithm. This rule calculates the number of eigenvectors that will be used as search vectors (I), such that the number of iterations is inversely proportional to the tightness of the problem. Moreover, the tightness increases the complexity of the problem of linear programming solved in every iteration. The graphics show small changes in the CPU time with respect to the number of constraints. This effect can be observed in Graph 1, where the difference between a problem with 30 constraints and another with only 5 constraints is on the order of a few seconds.

Graph 1: CPU time vs. tightness ratio.



Graph 2 clearly shows the dependence of the average CPU time on the dimension of the problem. This is due to the complexity of the larger problems, like those on the order of 500 variables. Naturally, the larger times on the order of 550 seconds are associated with those problems of high dimensions and with tightness ratios of high complexity (25 %).

Graph 2: Average CPU time vs. tightness ratio



3.2. Integrality GAP

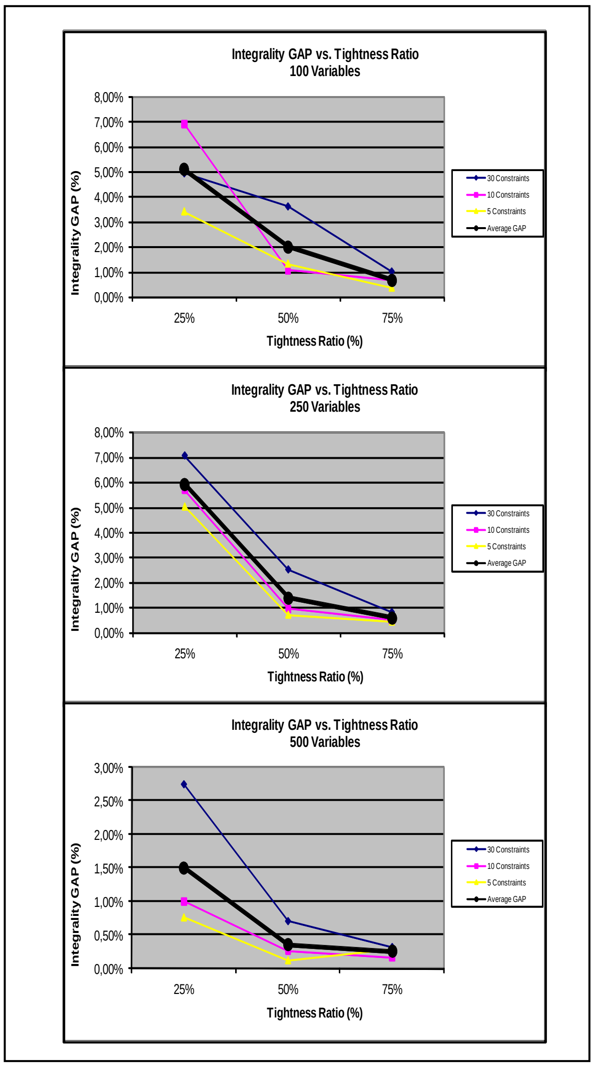
The quality of the results obtained using the heuristic algorithm can be evaluated using the Integrality GAP, which is defined as the percentage difference between the optimum solution and the approximate solution, according to the following formula:

$$GAP(\%) = 100 * \left(\frac{Sol_{IP} - Sol_{Heuristic}}{Sol_{IP}} \right).$$

Graph 3 depicts the behavior of the Integrality GAP with respect to the tightness ratio. Looking at those graphs, we can observe that the best results obtained by the heuristic are those obtained for the problems with tightness ratios of 50% and 75%. This may be due to the fact that higher complexity problems have fewer integer points. The obtained results stand out when the value of the tightness ratio is 75%, because the integrality GAP reaches 1%. These results also show that the integrality GAP increases with respect to the number of restrictions.

Observing the behavior of the integrality GAP in Graph 3, it is important to note that counterintuitively, the percentage difference between the approximate solution and the optimum value diminishes as the number of variables in the problem increases.

Graph 3: Integrality GAP vs. tightness ratio.



This can be explained by the fact that there are more eigenvectors used as searching directions, increasing the probability of finding an integer point.

4 Conclusion

Although the results of these experiments are preliminary, they show that the CPU time is improved in comparison with the CPU time of the optimal algorithms for integer programming. The heuristic algorithm finds good solutions for large problems in terms of the integrality gap.

For problems of 500 variables, the average integrality gap is 0.71% and the average CPU time is 352.81 seconds. In contrast, for smaller problems, the obtained results have an excessively large Integrality gap. For example, for problems of size 250 variables, the average integrality gap is 2.63% and the average CPU time is 39.34 seconds. The CPU time grows exponentially with the complexity of the instances of integer programming. The average integrality gap obtained for problems of 500 variables is on the order of the average number of components of the vector of the objective function. This shows that our solution is lesser aside in average, in one article.

This type of model is known to be very difficult to use in real situations because the CPU time for the optimal solution is prohibitive. For this reason, this heuristic algorithm appears to be very useful because for problems of 500 variables; in the worst cases, it takes about ten minutes, and the integrality gap is 2.75%. All the results reported here were obtained using MATLAB in Windows Vista and a domestic PC. The heuristic developed here can be improved using other solvers and more powerful computers. In the future, this work can be extended to other models in the context of integer programming.

References:

[1 J. E. Beasley OR-library, "*Multi- dimensional knapsack problems*", http://people. brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html

[2] S. Martello and P. Toth., "*Knapsack Problems: Algorithms and Computer Implementations*", Series in Discrete Mathematics and Optimization, Wiley Interscience. 1990.

[3] B. Spille and R. Weismantel, Primal Integer Programming In: K. Aardal, G. Nemhauser and R. Weismantel, Discrete Optimization. Elseiver Science. Vol.12, 2005 , pp.245-274.

[4] P. Chu and J. Beasley, A genetic algorithm for the multidimensional knapsack problem, Journal of Heuristics, Vol.4, 1998, pp.63–86.

[5] P. C. Gilmore and R. E. Gomory, The theory and computation of knapsack functions, Operations Research, Vol.14, 1966, pp.1045–1075.

[6] I. Derpich and J. Grandón, Some experimental results from a comparison between analytical and pseudo-Chebyshev center, **In progress.**