Accelerating the B&B algorithm for integer programming based on flatness information: an approach applied to the multidimensional knapsack problem

Ivan S. Derpich[1,†] **and Juan M. Sepúlveda**[1]

[1]*Industrial Engineering Department, Universidad de Santiago de Chile*
3363 Av.Estación Central, Santiago, Chile
E-mail: ⟨{ivan.derpich, juan.sepulveda}@usach.cl⟩

Abstract. This paper presents a new branching rule based on the flatness of a polyhedron associated to the set of constraints in an integer linear programming problem. The rule called Flatness II is a heuristic technique used with the branch-and-bound method. The rule is concerned with the minimum integer width vector. Empirical evidence supports the conjecture that the direction with the highest value of the vector's components indicates a suitable branching direction. The paper provides theoretical results demonstrating that the columns of the matrix A corresponding to a set of constraints Ax≤b may be used to estimate the minimum integer width vector; this fact is used for constructing a new version of the branching rule as was reported in a previous paper by the authors. In addition, the new rule uses a branching direction that chooses the child node closest to the integer value (either up or down). Thus, it uses a variable rule for descending the tree. Every time a new sub-problem is solved, the list of remaining unsolved sub-problems is analyzed, with priority given to those problems with a minimum objective function value estimate. The conclusions of the work are based on knapsack problems from the knapsack OR-Library. From the results, it is concluded that the new rule Flatness II presents low execution times and minimal number of nodes generated.

Keywords: integer programming, branch-and-bound method, branching rule, algorithm efficiency

Received: September 29, 2016; accepted: March 10, 2017; available online: March 31, 2017

DOI: 10.17535/crorr.2017.0008

1. Introduction

Mixed integer programming (MIP) is composed of a linear objective function, linear constraints and a set of variables where some or all are restricted to integers or binaries. The most popular method to solve these kinds of problems is the branch and bound algorithm (B&B), and is improved by using cuts (branch and

† Corresponding author

cut) and different heuristics such as local search, simulated annealing, Tabu search or reformulation techniques such as the Lagrangian relaxation, decomposition algorithms, column generation, among others. The B&B algorithm selects a node to be explored and solves a new problem as a relaxed linear programming (RLP) problem, where at least one of the integer variables has a non-integer solution. Thus, a search tree is generated when branching with these candidate variables. There are two aspects in browsing the search tree that will be considered in this work:

1) The branching strategy (i.e., a rule to select variables),
2) The branching direction (i.e., either the right or left node).

The most frequent approach in the B&B algorithm is to first select the variable for branching, and then to choose separately the branching direction using a fixed rule, for example, always using the left child node. In this classical approach, the method to solve the live nodes is determined in these two steps, that is, the variable and direction. This paper develops a branching rule based on the flatness of a polyhedron. The basis of the rule is a minimum integer width vector. In a non-empty closed polyhedron $K \subset \Re^n$ the width of this vector $d \in \mathbb{Z}^n$ is given by [1]:

$$w_d(K) = \min_d \left\{ \max_d \{d^T x : x \in K\} - \min_d \{d^T x : x \in K\} \right\} \quad (1)$$

To obtain the branching direction, the new rule is implemented by choosing the child node closest to the integer value (either up or down). Thus, instead of a fixed rule for descending the tree, a variable rule is used. A method of this type is known as informed search. This paper proposes a new rule for the branching variable, based on the flatness of a polyhedron. The flatness is estimated by the sum of the columns in matrix A corresponding to the set of constraints $A\,x \leq b$. The new rule acts in an integrated manner; that is, it combines the strategy for variable selection with the strategy for branching, along with the method for visiting the nodes in the waiting list. The proposed rule analyzes each candidate variable for branching by selecting the variable with the column of highest sum. It then uses the rounding fraction (upper or lower) to branch towards the direction with the smallest fraction. Moreover, each time a sub-problem is solved and a solution is obtained, the list of sub-problems is reviewed continuing with the sub-problem with lowest value in the objective function (in case of minimization). In the process, each time a new value is obtained from pruning, the list of live nodes is reviewed and the sub-problems with worst values are eliminated. The specific problem studied is the multidimensional knapsack problem (MKP), as in (2).

$$\left\{ \max c^T x : Ax \leq b, x \in \{0,1\}^n, a_{ij} \geq 0, b > 0 \right\} \quad (2)$$

2. The proposed branching rule: Flatness II

The branching rule developed in this paper is based on the geometry of a polyhedron associated with an integer problem. The rule is called Flatness II, named after the work by Derpich and Vera [1]. In their work, the branching rule utilizes the eigenvalues of a matrix corresponding to an inscribed Dikin ellipsoid [2]. However, experimenting with the method in [1] shows that is not adequately efficient as it requires long CPU processing times in calculating the center point of the ellipsoid and the eigenvalues of the matrix. The new proposed method uses the columns of matrix A. The development considers two aspects: i) a rule for branching, and ii) a rule for selecting the child node to be branched. The branching rule proposed in this paper is based on the flatness direction and is to be constructed in two steps as given below.

2.1. The Flatness II rule

The integer width of the polyhedron associated with the integer programming problem is a good start for obtaining branching priorities. The idea is to move first in the direction where the polyhedron is flatter. Then, the following construction is necessary:

Let $K \subset \Re^n$ be a non-empty closed set and let $d \in \Re^n$ be any vector. The width of K along d is given by (1) as shown above. And the integer width of K is defined as:

$$w(K) = \min_{d \in \mathbb{Z}^n \setminus \{0\}} w_d(K) \tag{3}$$

Thus, any d that minimizes $w_d(K)$ is called the direction of minimal integer width of K.

Theorem 1 (Kinchin's Flatness Theorem [3]). Let n be the dimension of the set K, then there exists a function $w(n)$ depending only on the dimension, such that if $K \subset \mathbb{R}^n$ is convex and $w(K) > w(n)$, then set K contains an integer point.

The best known bound for $w(n)$ is $O\left(n^{3/2}\right)$ and it is conjectured that $w(n) = \Theta(n)$. For an interior ellipsoid of a polyhedron, for example a Dikin ellipsoid, the flatness direction can be computed by solving the shortest vector problem (SVP) in a base of the lattice $\mathcal{L}\left(\left(Q^{1/2}\right)^T\right)$, where $Q = A^T D(x_0)^{-2} A$, and $D = \mathrm{diag}\left(b_1 - \alpha_1^T x_0, b_2 - \alpha_2^T x_0, \ldots, b_m - \alpha_m^T x_0\right)$, where m is the number of constraints of the problem and α_i the *i-th* row of the matrix A.

In the MKP, if we relax the constraint $x \in \{0,1\}^n$ and change it applying the constraint $0 \le x_j \le 1, \forall j = 1, .. n$, the feasible region becomes the unit hyper cube (UHC). And if we suppose that the relaxed polyhedron K has at least one integer point, then we can approximate the center of the polyhedron K.

Finally we have that $D = diag(h_1, h_2, \ldots, h_m)$ and $h_1 \leq 0.5; \ldots; h_m \leq 0.5$; by bounding D we have that $D = diag\left(\frac{1}{2}, \ldots, \frac{1}{2}\right) = \frac{1}{2} diag(1, \ldots, 1) = \frac{1}{2} I$ and then $Q = A^T D^{-2} A = 4A^T A$. The base for the lattice $\mathcal{L}$ is a function of the matrix A.
Computing the direction of minimum integer width of K is an NP-hard problem. We establish a set of algebraic developments and generate several forms for computing a score to decide on the branching of variables. Hence, the rule proposed in this work was first inspired by the flatness theorem of Kinchin and the particular condition of the MKP. In the MKP, we will search for a way to calculate the minimum integer width vector, knowing that this vector can be approximated by the shortest vector problem (SVP) and using a lattice basis that depends only on matrix A. We find the value b_i/a_{ij}, as shown in Figure 1, which corresponds to the point of intersection of constraint i with the Cartesian axis associated with x_j. We then establish a relationship between the minimum integer width vector and the term b_i/a_{ij}which is not exact, but a heuristic relationship. This is still experimental work with unresolved issues, which occurs often in integer programming.

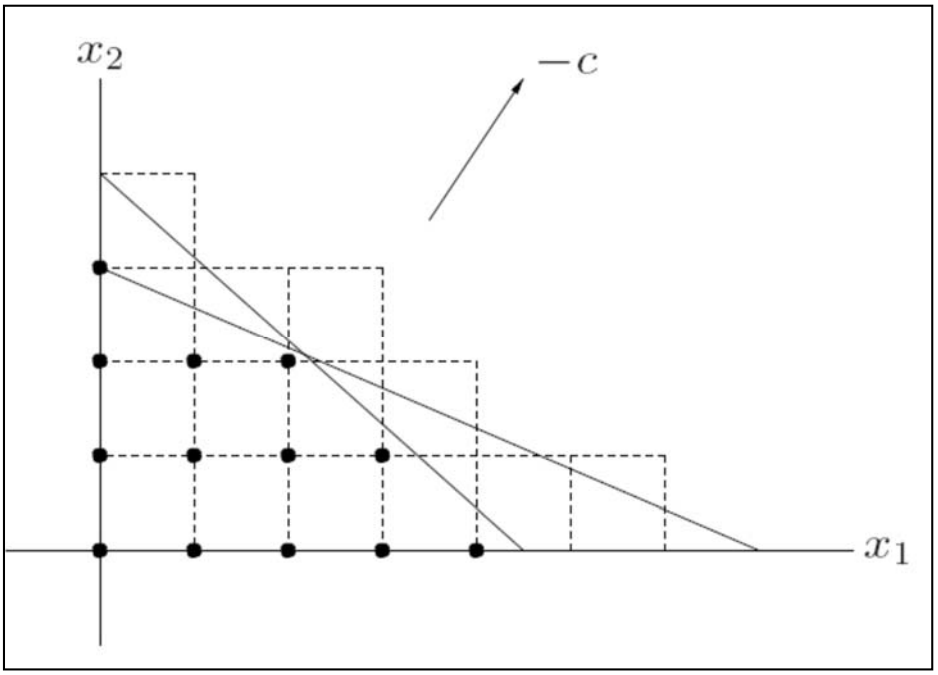

Figure 1: *Points of intersection between constraints and coordinate axes*

Let us call this value k_{ij} and $\tilde{k}_j$ the mean value of the intersecting points of constraint i w.r.t. the Cartesian axis j. Let F be the set of integer variables that has fractional values in the last linear programming optimum solution.

$$\tilde{k}_j = \sum_{i=1}^{m} \frac{b_i}{a_{ij}}, \quad \forall j = 1 \dots n \tag{4}$$

Where, $x_j \in F$. Moreover, the value in (5) is considered a linear estimator of components j of the minimum integer width $\bar{d}_j$.

$$\bar{k}_j = \frac{1}{m} \sum_{i=1}^{m} \frac{b_i}{a_{ij}}, \quad \forall j = 1, \dots n \tag{5}$$

Where, $x_j \in F$. By assuming that b_i is approximately constant and by making $b_i = b$, it then holds that (6):

$$\bar{d}_j = \frac{1}{\bar{k}_j} = \frac{m \sum_{i=1}^{m} a_{ij}}{b}, \quad \forall j = 1 \dots n \quad \text{such that } x_j \in F \tag{6}$$

Given that b is the same value for all directions and m is constant, it leads us to the consideration that:

$$\hat{d}_j = \sum_{i=1}^{m} a_{ij}, \quad \forall j = 1 \dots n \tag{7}$$

The value in (7) is an estimator of the minimum integer width vector $\bar{d}_j$. Thus, the score $s(j)$ for each candidate variable is:

$$s(j) = \sum_{i=1}^{m} a_{ij}, \quad \forall j=1 \dots n \quad \text{such that} \quad x_j \in F \tag{8}$$

Where a_{ij} is the *i,j* component of matrix *A*. Variable x_j with a maximum *s(j)* value is selected for branching. The idea behind this rule is to branch first using the variable that geometrically corresponds to a dimension where matrix $\boldsymbol{A}$ has less density. Intuitively, this measure is to some extent similar to polyhedral flatness; however, it is different as it does not take into account the right-hand side of the constraints.

i) Rule for the branching direction:

Next, a decision is required as to which child to branch first; the selection proposed in this paper is based on the adjustment of minimum effort to the nearest integer. Let (9) be the fractions of upper and lower rounding, respectively.

$$f_j^+ = \lceil x_j \rceil - x_j \quad \text{and} \quad f_j^- = x_j - \lfloor x_j \rfloor \tag{9}$$

If (10) holds, then the left child is branched; that is, a sub-problem is solved by adding $x_j \leq \lfloor x_j \rfloor$. Otherwise, the right-hand child is branched, that is, the constraint $x_j \geq \lceil x_j \rceil$ is added. As indicated above, the idea behind this rule is to make a least effort adjustment.

$$f_j^+ > f_j^- \tag{10}$$

3. Comparison between the Flatness II rule and other rules found in literature

Next, this section provides a brief overview of branching rules and additional aspects regarding the efficiency of the B&B algorithm.

3.1. Branching rules

i) Most infeasible and least infeasible branching, where most infeasible branching chooses the fractional variable with the decimal part closest to 0.5, whereas least infeasible branching chooses fractional variables closer to the integer.

ii) Pseudocosts (PSC) was developed by Bénichou et al. [3]; this rule accumulates a history of the successes relating to variables that were already branched by comparing the candidates through the gain for each case:

$$\Delta_j^+ = \bar{c}_{Q_j^+} - \bar{c}_Q \quad \text{and} \quad \Delta_j^- = \bar{c}_{Q_i^-} - \bar{c}_Q \tag{11}$$

for upper rounding and lower rounding, respectively. Where $\bar{c}_{Q_j^+}$ and $\bar{c}_{Q_i^-}$ are the values of the objective function when solving the sub-problems $\bar{c}_Q$ upon adding the constraints $x_j \geq \lceil x_j^* \rceil$ and $x_j \leq \lfloor x_j^* \rfloor$, respectively; where x_j^* is the component j of the variable x^*, the optimal solution of the problem $\bar{c}_Q$.

$$\Psi_j^+ = \sum_{j \in S_j} \frac{\Delta_j^+}{f_j^+ \eta_j^+}; \qquad \Psi_j^- = \sum_{j \in S_j} \frac{\Delta_j^-}{f_j^- \eta_j^-} \tag{12}$$

Where η_j^+ is the number of these problems with upper rounding and η_j^- with lower rounding. The term f_j^+ and f_j^- were defined in (9).

$$j^* = score\left(f_j^- \Psi_j^-, f_j^+ \Psi_j^+\right) \tag{13}$$

There are different score functions; one used by Achterberg [5] [6] is the following:

$$j^* = (1-\mu)\min\{ f_j^- \Psi_j^-, f_j^+ \Psi_j^+ \} + \mu \cdot \max\{ f_j^- \Psi_j^-, f_j^+ \Psi_j^+ \} \quad (14)$$

where μ called the *factor score* is a number between 0 and 1 (utilized $\mu = 1/6$).

iii) Strong branching (STG) is one of the most widely used approaches, proposed by Applegate et al. [7] [8]. The algorithm carries out a stepwise forward search for each variable not complying integrality at the nodes. The computational solution searches the relaxed *LP* solution for each child, thus obtaining $\boldsymbol{D_i^s}$ $(\boldsymbol{S} = +, -)$.

$$j^* = \mu_1 \cdot min\{ D_j^-, D_j^+ \} + \mu_2 \cdot max\{ D_j^-, D_j^+ \} \quad (15)$$

Where $j*$ is the degradation progress of the child nodes, as suggested by Eckstein [9], where the parameters $\mu_1 = 4\ and\ \mu_2 = 1$ were empirically verified by Linderoth and Savelsbergh [10]. Moreover, if the forward search should continue 2^k times utilizing the degradation of the child nodes, it leads to lookahead branching for $\boldsymbol{\lambda}$= *4,8,..,*2^k, such that $k \in \boldsymbol{N}$. Thus, if *k=1*, the described algorithm will be full strong branching.

iv) Entropic branching (ENTR) aims to reduce the uncertainty (*entropy*) of current sub-problems; that is, quantifies the "uncertainty" of the partial solution. This principle borrows some definitions from information theory, and its main contribution is the notion of entropy. According to this theory, entropy is an additive for independent variables and defines the *entropy of a group of binary variables*, proposed by Andrew and Sandholm in [11] as the sum of the entropy over a set of probabilities corresponding to independent binary events.

v) Lookahead branching (LOOK) was proposed by Glankwamdee and Linderoth [12]. It estimates the impact of the current solution by considering in depth two child nodes and counting the number of potential nephews possibly evaluated if the variable *i* is chosen as a branching variable.

vi) Reliability branching (RELI) was introduced by Achterberg at al. [5], and the rule applies strong branching on the upper side of a tree to depth of $d \cong \eta_{rel}$. The algorithm calculates $s(j) = score\left(f_j^- \Psi_j^-, f_j^+ \Psi_j^+\right)$ for all candidate variables and sorts them in decreasing order. Then, for all candidate variables *j*, such that $\min\{\eta_j^-, \eta_j^+\} < \eta_{rel}$, the subproblems S_j^- *y* S_j^+ are resolved as relaxed *LP* problems, Δ_j^+, Δ_j^- and the pseudocosts Ψ_j^+ and Ψ_j^- are updated. Finally, the candidate variable with the highest score is chosen. If η_{rel} is equal to zero, it coincides with pure pseudocosts branching, otherwise it tends to ∞ and converges towards strong branching. Additionally, if $\lambda = \infty$, strong branching becomes full strong branching.

vii) The hybrid rule Flatness II and Pseudocosts (FLPS) is a rule that combines the Pseudocosts rule and Flatness II as proposed by Derpich and Macuada in [12]. It is known that Pseudocosts rule uses the information from other branching to calculate the estimated cost of a new branch. Given that at the beginning of the process there are no branches, then there is no information and therefore this rule is almost a random decision and if the variable to be branched is badly chosen, the process will follow a wrong path in the search tree. Due to the latter, the hybrid rule starts by first using the Flatness II, and when the process has reached a certain branching history, it then flips to the Pseudocosts rule.

viii) The flatness rule (FLAT) is the original rule developed in [1]. Here, ellipses based on a logarithmic barrier function are used for the polyhedron. Let $P = \{x: \alpha_i x \leq b_i, i = 1,..,m\}$, be a pair of concentric ellipses defined by the form $E = \{x \in \mathbb{R}^n: (x - x_0)^n Q(x - x_0) \leq 1\}$ and $E' = \{x \in \mathbb{R}^n: (x - x_0)^n Q(x - x_0) \leq \gamma^2\}$, where x_0 is the center of the ellipses, such that $E \subset P(d) \subset E'$ and $Q = \nabla^2 \rho(x_0) = A^T D^{-2}(x_0) A$ with $D(x_0) = diag(b_i - \alpha_i^T x_0)$. It can then be shown that the ellipsoid constructed using Q satisfies the required properties and $\gamma = m + 1$. In [1], the shortest axis of the ellipse was used as an approximation of the minimum integer width direction. Accordingly, it appears reasonable to guide the search strategy using the branch and bound algorithm in terms of an estimated shape of the polyhedron, a measure which is reflected by the minimum integer width vector. Given the complexity in computing it, [1] proposed a selection rule for the branching variable based on vectors corresponding to the principal axes of the Dikin ellipse associated to center point. In general, although the smallest axis will differ from the minimum integer width vector, we expect to recover from it significant information on the spatial orientation of the polyhedron with respect to the integer lattice. As indicated above, Lenstra's algorithm takes explicit advantage of this information. We will use the vector corresponding to the shortest axis of the Dikin ellipse to define priorities for the variables. These priorities will be incorporated into the branch and bound. However, we notice that it might be meaningless to search in some of the directions defined by the shortest axis if they are too small. This last fact is an indication of the polyhedron being thin in one direction. In that case, finding many more integral points in the orthogonal directions is more likely as more integral coordinates exist in those directions. The following proposition justifies the claim:

Proposition 1.
Let $(\beta_1, \ldots, \beta_n)$ *be the vector corresponding to the shortest semi-axis of the Dikin ellipse constructed with point* x^0 *as the center .Let* $\delta = \min\{|\beta_j|: j = 1, \ldots, n\}$. *Let* $B(x^0, \delta)_\infty$ *be the ball, in the* L_∞*norm, of radius* δ *with the center at* x^0. *Then, if* $\delta < 0.5$ *and* x^0 *is the center of the unit hypercube, there is no nonzero integral point in the interior of* $B(x^0, \delta)_\infty$.

Proof. This is a simple geometric fact.

Now we are ready to define the priority vector. Let $(\beta_1, \dots, \beta_n)$ be the coordinates of the shortest axis of the Dikin ellipsoid. Let $\Theta = \{\beta_j: |\beta_j| > 1/2\}$. From the previous result, the coordinates in this set correspond to the directions in which it is more probable to find integer coordinates for points contained in the polyhedron. We give priorities to the variables, from 1 to n (with 1 being the highest priority) in the following way: Let $k = \arg\max_j\{|\beta_j|: \beta_j \in \Theta\}$. Then, we assign $\alpha_k = 1$. Now, let $l = \arg\max_j\{|\beta_j|: \beta_j \in \Theta - \{\beta_k\}\}$. We assign $\alpha_l = 2$. Upon repeating this process, we complete assigning priorities to the variables for the components of the set Θ. For components not in Θ, we assign a priority of zero which, when implementing the software, is equivalent to assigning the last priority for branching.

Algorithm set-priority
INPUT: matrix A in $\mathbf{R}^{m\times n}$ and vector $b \in \mathbf{R}^m$
OUTPUT: A priority vector α for the branch and bound procedure.
1. Solve the linear program: $\max t$, subject to $Ax + te \le b$, $t \ge 0$
where e is the vector $(1, \dots, 1)^{\mathrm{T}}$. Let x_0 be the minimizer. This point is in the interior of the polyhedron, in fact, it is the center of the largest sphere contained in the polyhedron.
2. Let $Q = \nabla^2\rho(x_0) = A^T D^{-2}(x_0)A$ where $D(x_0) = diag(b_i - \alpha_i^T x_0)$
3. Let $(\beta_1, \dots, \beta_2)$ be the shortest semi-axe of $E = \{x \in \mathbb{R}^n: (x - x_0)^n Q(x - x_0) \le 1\}$
4. Let $\Theta=\{\beta j:|\beta j|>1/2,j=1,\dots,n\}$.
5. Let $p = 1$, $S = \Theta$.
6. **Repeat until** $S = \emptyset$
Let $k = \arg\max_j\{|\beta_j|: \beta_j \in S\}$.
$\alpha_k = p$
$S = S - \{\beta_k\}$, $p = p + 1$.
end repeat
7. The priorities for the remaining components are set to zero.

4. Experimental results

The design and measurements were executed in a personal computer using an Intel Core i7 ™, 2.3 GHz, Windows 7 Pro™ operating system, and 8 Gb RAM. The design was conducted entirely in *MATLAB™* and its associated compiler. Our proprietary code named *BeBe* and programmed in the *MATLAB™* language was developed for the project which works with sparse matrices. The solver manager OPTI *Toolbox™* used a series of solvers linked to the *MATLAB™* tools.

In particular, the *MOSEK™* solver for LP problems was used, which uses interior point methods for finding solutions to problems.
It is known fact that most scientific publications utilize commercial solvers during development, as they achieve shorter processing times, and that interpreted code executes slower that compiled code in machine language. However, the number of nodes identified for solving the problem should be equivalent in both cases. This strategy was an advantage since using our proprietary code ensured that no other techniques became a hindrance, such as preprocessing, specific heuristics, cover inequalities, among others, which may have affected evaluation of the desired result. Our own clean code ensured that only the utilized branching rules influenced the evaluated variables, i.e., CPU time and visited nodes. The developed algorithm works with data in sparse matrix format and works directly with the MPS format. Next, the paper will present the results divided into a set of tables corresponding to the knapsack OR–Library [13]. The main feature for mknap-01 and mknap-02 is variability in multidimensional problems. The set of respective problems mkanpcb1 belong to the 30 problems of the same dimension; however, they are highly complex due to the intensity in the branching. The compared algorithms and their abbreviation are as follows:

- PSC: Pseudocosts
- STG: Strong branching
- ENTR: Entropic branching
- RELI: Reliability branching
- FLAT II: New rule proposed
- FLAT: Flatness polyhedron
- LOOK: Lookahead branching
- FLPS: Flatness II/ Pseudocosts

The results corresponding to the eight strategies reviewed in the literature are shown below. Besides the new branching rule, Flatness II was also tested, using a dynamic method for selecting the child node to be branched.

N°	Size	PSC	FLAT II	STG	FLAT	ENTR	LOOK	RELI	FLPS
1	6x10	11	11	10	10	10	11	11	10
2	10x10	20	12	17	15	13	14	17	13
3	15x10	134	62	112	112	167	158	108	122
4	20x10	87	40	69	50	172	188	148	43
5	28x10	158	98	398	386	483	409	386	378
6	39x5	271	51	181	211	545	946	496	280
7	50x5	428	253	534	489	15566	1273	1273	522
Average		158	75	180	182	2422	669	348	195

Table 1: *Comparison of nodes visited in the branching rules relating to file mknap-01.txt*

Table 1 shows the results corresponding to the number of visited nodes for the set of problems relating to file mknap-01, and incorporates the results obtained for different branching strategies. The minimum values are shown in bold. The results show that Flatness II rule used the fewest nodes in five out of eight cases. Table 2 shows the CPU time for the set of problems relating to file mknap-01.txt, Flatness II is one of the rules using least time and has the shortest time in all cases, although in two cases it tied with Pseudocosts.

N°	Size	PSC	FLAT II	STG	FLAT	ENTR	LOOK	RELI	FLPS
1	6x10	**0.03**	**0.03**	0.05	0.05	0.04	0.06	0.07	0.05
2	10x10	**0.02**	**0.02**	0.03	0.03	0.03	0.06	0.03	**0.02**
3	15x10	0.13	**0.08**	0.15	0.14	0.22	0.51	0.13	0.12
4	20x10	0.11	**0.06**	0.14	0.11	0.3	0.75	0.23	0.08
5	28x10	0.16	**0.14**	0.5	0.45	0.7	1.13	0.47	0.44
6	39x5	0.26	**0.08**	0.37	0.35	1.10	6.2	0.57	0.32
7	50x5	0.51	**0.42**	1.35	0.96	45.93	21.04	1.66	0.75
Avera-ge		0.2	**0.1**	0.4	0.3	6.9	4.3	0.5	0.2

Table 2: *Comparison of CPU times in the branching rules relating to file mknap-01.txt*

Table 3 shows the results presented corresponding to the number of visited nodes for the set of problems of the file mknap-02 in which the results obtained for the different branching strategies are presented. The minimum values are presented in bold. The results show that the rule FLAT II (Flatness II) is the one that minimize in average the number of visited nodes. The second rule is STG (Strong Branching) followed by FLAT (the former Flatness rule, as described in section 3).

	Size	PSC	FLAT II	STG	FLAT	ENTR	LOOK	RELI	FLPS
1	60x30	1557	1588	**499**	1895	1048	813	1271	1471
2	60x30	1833	**1247**	1358	5417	7892	5162	2637	4495
3	28x2	156	194	132	133	147	214	**117**	156
4	28x2	51	52	**50**	52	130	**50**	**50**	52
5	28x2	211	**78**	79	78	69	198	74	194
6	28x2	58	**25**	41	25	99	27	106	25
7	28x2	80	**22**	22	22	22	22	22	22
8	28x2	141	**108**	119	**108**	115	110	109	109
9	105x2	264	**83**	239	194	280	445	453	353
10	105x2	6616	**437**	1465	1166	7434	6712	1268	6746

11	30x5	187	**117**	118	180	202	186	215	167
12	30x5	115	**62**	90	86	130	130	195	83
13	30x5	73	**41**	58	52	270	226	113	116
14	30x5	33	**31**	33	31	33	31	131	33
15	30x5	41	**40**	67	41	60	45	43	41
16	40x5	390	**105**	272	142	417	439	304	273
17	40x5	323	**141**	228	139	719	230	511	343
18	40x5	111	**84**	181	83	148	137	195	118
19	40x5	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**
20	50x5	314	**144**	271	250	403	453	344	317
21	50x5	345	**46**	77	46	125	126	118	77
22	50x5	99	**69**	100	75	178	107	167	124
23	50x5	517	**90**	217	57	137	138	61	90
24	60x5	92	**62**	70	70	116	109	70	68
25	60x5	179	**112**	183	185	188	215	231	182
26	60x5	1833	**1358**	2247	2417	7892	5162	2637	2595
27	60x5	194	156	132	133	147	214	**117**	156
28	70x5	131	53	51	52	130	52	**50**	**50**
29	70x5	211	**78**	109	78	69	198	71	194
30	70x5	41	**25**	58	**25**	99	27	106	**25**
31	70x5	**22**	**22**	80	**22**	**22**	**22**	**22**	**22**
32	80x5	119	**108**	115	**108**	115	110	109	109
33	80x5	264	**83**	239	194	280	445	453	353
34	80x5	6616	**437**	1465	1166	7434	6712	1268	2746
35	80x5	187	**117**	180	118	262	186	215	167
36	90x5	115	**62**	115	90	130	130	195	83
37	90x5	73	**41**	103	58	102	270	226	116
38	90x5	33	**31**	33	31	33	32	131	33
39	90x5	67	**40**	41	40	41	40	45	41
40	39x5	390	**105**	272	242	417	439	304	273
41	27x4	323	**141**	228	239	719	230	511	343
42	34x4	111	**83**	84	181	148	137	118	195
43	29x2	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**
44	20x10	344	**250**	271	250	403	453	344	317
45	40x30	345	**46**	77	47	125	126	118	77
46	37x30	99	**69**	100	135	178	107	167	124
47	28x4	517	**57**	90	217	137	138	61	90
48	35x4	92	**68**	70	82	116	109	70	69
Average		568	**204**	243	432	842	672	358	646

Table 3: *Comparison of nodes visited in the branching rules relating to file mknap-02.txt*

Regarding the visited nodes, the best rule was Flatness II followed very closely by strong branching. This competitiveness trend between both, relating to the number of nodes visited, is a characteristic that repeats in the following experiment with another group of instances. In third place is the Reliability rule with an average of 358 nodes. Tables 4 shows the CPU times of the experiments performed with the instances taken from the file mknap-02 in the OR-Library.

	Size	PSC	FLAT II	STG	FLAT	ENTR	LOOK	RELI	FLPS
1	60x30	**5.44**	5.54	7.09	11.58	13.32	57.48	6.6	5.66
2	60x30	16.19	**6.26**	8.05	25.84	83.13	229.94	8.71	13.1
3	28x2	**0.17**	0.23	0.21	0.18	0.23	0.82	0.19	0.19
4	28x2	**0.06**	0.07	0.08	0.16	0.19	0.20	0.11	0.08
5	28x2	0.13	0.13	0.14	**0.12**	0.24	0.79	0.15	0.24
6	28x2	0.06	**0.04**	**0.04**	0.07	0.17	0.17	0.17	0.05
7	28x2	**0.03**	0.04	**0.03**	**0.03**	**0.03**	0.07	0.07	0.10
8	28x2	0.13	0.17	0.13	**0.11**	0.14	0.32	0.15	0.12
9	105x2	0.43	**0.16**	0.56	0.41	0.65	2.60	0.78	0.57
10	105x2	10.68	**0.81**	3.68	2.47	23.75	49.35	2.76	11.30
11	30x5	0.14	0.15	0.26	0.33	0.40	1.22	0.35	0.23
12	30x5	**0.07**	0.14	0.14	0.13	0.22	0.50	0.31	1.20
13	30x5	0.10	**0.06**	0.14	0.22	0.67	0.15	0.21	0.19
14	30x5	**0.05**	**0.05**	0.07	0.06	0.07	0.24	0.25	0.11
15	30x5	**0.07**	0.08	0.08	0.08	0.08	0.21	0.11	0.41
16	40x5	0.52	**0.16**	0.65	0.47	0.88	2.44	1.50	0.41
17	40x5	0.43	**0.34**	0.39	0.43	0.75	1.66	0.83	0.54
18	40x5	0.14	**0.13**	0.23	0.34	0.34	0.69	0.23	0.3
19	40x5	**0.04**	**0.04**	**0.04**	**0.04**	0.05	0.11	0.15	0.05
20	50x5	**0.41**	0.53	0.71	0.49	0.92	2.58	0.68	0.48
21	50x5	0.11	0.14	0.13	**0.10**	0.26	0.77	0.27	0.15
22	50x5	0.14	**0.11**	0.22	0.28	0.45	0.72	0.38	0.24
23	50x5	0.13	**0.11**	0.26	0.18	0.26	0.96	0.22	0.15
24	60x5	0.16	**0.11**	0.26	0.18	0.26	0.96	0.22	0.15
25	60x5	0.29	**0.20**	0.46	0.40	0.43	1.31	0.52	0.34
26	60x5	**6.26**	8.05	16.1	25.8	83.1	229.9	8.71	13.1
27	60x5	0.23	**0.17**	0.21	0.18	0.23	0.82	0.19	0.19
28	70x5	0.16	**0.06**	0.08	0.07	0.19	0.2	0.11	0.08
29	70x5	0.13	**0.14**	0.14	0.12	0.13	0.79	0.15	0.24
30	70x5	0.06	**0.04**	0.07	0.04	0.17	0.17	0.17	0.05
31	70x5	0.03	**0.03**	0.03	0.03	0.03	0.07	0.17	0.04

32	80x5	0.13	**0.11**	0.13	0.17	0.14	0.32	0.15	0.12
33	80x5	0.43	**0.16**	0.56	0.41	0.65	2.6	0.78	0.57
34	80x5	10.68	**0.81**	3.68	2.47	23.75	49.35	2.76	11.3
35	80x5	0.14	**0.15**	0.26	0.33	0.4	1.22	0.55	0.23
36	90x5	0.17	**0.14**	0.14	0.13	0.22	0.5	0.41	1.20
37	90x5	0.06	**0.1**	0.14	0.22	0.67	0.15	0.21	0.19
38	90x5	0.06	**0.05**	0.07	0.06	0.07	0.24	0.25	0.11
39	90x5	0.08	**0.09**	0.07	0.08	0.08	0.21	0.21	0.41
40	39x5	0.52	**0.16**	0.65	0.47	0.88	2.44	0.50	0.41
41	27x4	0.43	**0.34**	0.39	0.43	0.15	1.66	0.83	0.54
42	34x4	0.14	**0.13**	0.23	0.34	0.34	0.69	0.23	0.30
43	29x2	0.04	**0.04**	0.04	0.04	0.05	0.11	0.10	0.05
44	20x10	0.41	**0.53**	0.71	0.49	0.92	2.58	0.68	0.48
45	40x30	0.11	**0.54**	0.13	0.10	0.26	0.77	0.27	0.15
46	37x30	0.14	**0.11**	0.22	0.28	0.45	0.72	0.38	0.24
47	28x4	0.13	**0.18**	0.51	0.12	0.31	0.80	0.21	0.17
48	35x4	0.16	**0.12**	0.26	0.18	0.26	0.96	0.22	0.15
Av.		1.18	**0.59**	1.03	1.61	5.03	13.61	0.92	1.39

Table 4: *CPU-Time for the set of problems of the mknap-02*

The best rule in terms of CPU time was by far the Flatness II rule. Here, the rule Pseudocosts had a very high CPU time compared to what was expected. The other rules had longer computational processes, where the Entropic rule was especially slow. Table 5 shows the number of nodes visited for the set of problems relating to mknapcb-01. Tables 6 shows experiments performed using the instances taken from the same file mknap-02 of the OR-Library, but now showing results for visited nodes. The best rule in this case was Strong branching, and is not surprising as it generated more information due to solving a larger number of sub-LP problems. Consequently, it is slower than the other rules. However, the difference with Flatness II in terms of nodes is 39852 nodes in the 30 problems solved, representing an average of 1328 nodes per problem, that is, using Flatness II versus Strong branching. Table 6 shows CPU times for the set of problems from mknapcb-01 in seconds.

In terms of processing time, the best rule was Flatness II, and the second best was Pseudocosts by 37%. The third best rule was FL/PS, a rule that combines both Flatness II and Pseudocosts, using Flatness II for the first 20 iterations and Pseudocosts for the remaining. This was unexpected given that in the other experiments with different groups of instances, the FL/PS rule based on this benchmark appeared more distant than the best rules.

	PSC	FLAT II	STG	FLAT	ENTR	RELI	FLPS
1	588381	324280	**139991**	254726	546946	1078617	611384
2	264158	171439	145417	**139624**	364005	703531	175685
3	324359	316426	**168767**	205934	319263	979461	379138
4	964082	601654	**578440**	1220753	1703279	3442123	2059026
5	212887	208628	**165941**	344996	400403	696683	526471
6	291613	**113822**	152765	150751	304693	391331	223764
7	**134810**	44861	207494	220550	598252	396170	319995
8	129313	160300	**99974**	186669	165984	517715	295378
9	241756	271505	**123527**	417345	295741	1093448	755203
10	735438	**223578**	235629	536155	487905	1362273	1073853
11	205735	**68259**	166865	167824	505815	624915	213552
12	197840	**131542**	133426	398402	294152	426365	761022
13	1391085	915001	**498013**	1250885	1344622	2408570	2131063
14	378781	317403	**222035**	292024	465836	1105971	490193
15	419797	547862	412136	**370976**	794080	888985	378254
16	**41087**	148398	166495	73497	290279	283302	76198
17	10114	**6414**	38784	47624	307866	152130	54602
18	818034	461762	**246821**	406525	773227	1025309	578438
19	972162	179190	**137278**	285492	477214	863642	315320
20	252301	191252	**92479**	179493	380977	511477	228402
21	85236	50951	80105	**45814**	229241	271838	46322
22	167063	**49064**	142109	112138	465920	384785	89981
23	113829	100445	**77209**	106397	278455	268344	122413
24	126435	170148	118677	136646	474774	347865	142646
25	**50249**	62639	61747	65518	449974	285441	66820
26	210992	147275	**121214**	161497	328954	507728	251729
27	58028	**52778**	76365	60047	835363	497729	88106
28	78758	89968	**46067**	91884	360461	213979	93901
29	**61990**	80291	62824	**127933**	489723	692423	89854
30	131577	**127878**	220874	**198412**	522641	615543	197293
Average	321930	211167	**171316**	275218	508535	767923	427867

Table 5: *Number of nodes visited for the set of problems relating to mknapcb-01*

	PSC	FLAT II	STG	FLAT	ENTR	RELI	FLPS
1	917	**581**	706	833	2927	1397	940
2	403	280	677	431	1956	874	**262**
3	**496**	520	808	639	1690	1248	567
4	1471	**977**	3044	3970	9141	4444	3111
5	**330**	343	820	1128	2149	905	801
6	440	**186**	709	472	1551	484	335
7	202	**73**	921	676	2891	496	475
8	**201**	262	456	567	844	653	436
9	**371**	446	613	1278	1533	1359	1132
10	1125	**366**	1135	1701	2602	1707	16217
11	301	**112**	666	485	2391	718	308
12	288	**215**	554	1105	1365	490	1092
13	2081	**1496**	2354	3686	6801	2848	3126
14	551	**519**	958	828	2174	1240	701
15	**612**	892	1779	1034	3637	983	538
16	**61**	247	732	206	1386	327	110
17	15	**11**	138	128	1374	169	75
18	1161	**750**	1131	1130	3581	1129	803
19	1424	**292**	570	810	2300	951	454
20	379	**311**	388	523	1846	577	331
21	112	83	287	122	827	280	**63**
22	221	**81**	494	265	1665	398	111
23	155	**163**	293	277	1031	288	166
24	**174**	278	471	358	1836	367	194
25	**66**	103	222	164	1628	290	86
26	288	**242**	504	420	1287	534	332
27	**79**	87	266	153	3154	515	120
28	**105**	146	167	222	1299	225	120
29	**81**	129	226	310	1773	703	114
30	**180**	208	858	539	2024	637	264
Average	476	**346**	764	815	2355	908	626

Table 6: *CPU-Times for the set of problems relating to the mknapcb-01 file*

5. Conclusions

Based on the developments in this research and the obtained results, our conclusion is that the actions taken to devise a new branching rule based on the polyhedron, as implemented in the B&B algorithm were adequate. The FLAT II rule developed in this paper aimed to improve the CPU time and the memory size

used in the B&B algorithm. These factors are important in resolving real-world problems, for instance, in industry and business, especially in large scale problems. A good example of these factors is found in areas where integer programming has assumed the main role, such as telecommunications networks, due to the large number of variables taken into account in decision making. The size makes many problems slightly intractable, as for example a network with one hundred nodes using a special linear formulation of the p-hub type. The formulation requires five hundred integer variables with a resolution that may take hours using even efficient commercial software. In cases like these, efficient strategies should exist for branching variables, directing and handling the list, so that the B&B capacity is improved in order to obtain solutions within reasonable periods of time. The conclusion of the work is based on the knapsack problems from the knapsack OR-Library. These have integer variables 0 and 1, and are commonly used for testing as they provide non-polynomial complexity. Based on the results, the conclusion is that the devised rule presents low execution times and a minimal number of generated nodes. Future work in the proposed design should include testing the algorithm using other libraries with a variety of problems.

Acknowledgements

The authors are very grateful to DICYT (Scientific and Technological Research Office), Project Number 061317DC and the Industrial Engineering (IE) Department at the University of Santiago of Chile for supporting this work. In addition, my appreciation is also directed to Mr Danny Chamaca, a postgraduate student studying for a master of science degree in IE, who also worked on implementing the model.

References

[1] Derpich, I., Vera, A. (2006), Improving the efficiency of the Branch-and-Bound algorithm for integer programming based on “flatness” information. European Journal of Operational Research, v174, I1, pp. 92-101.

[2] Dikin I. (1967). Iterative solutions of linear and quadratic programming problems, Soviet Mathematic Doklady, 8 pp. 674–675.

[3] Kinchin, A., (1948), A quantitative formulation of Kronecker’s theory of approximation. Izvestiya Akademii Nauk SSR Seriya Matematika 12 113-122 (in Russian).

[4] Benichou, M.; Gauthier, J.M.; Girodet, P.; Hentges, G.; Ribiere, G.; Vicent, O. (1971). Experiments in mixed-integer programming, Math Programming, 1, p76-94.

[5] Achterberg, T; Koch, T; Martin, A. (2005). Branching rules revised. Operations Research Letters, 33, p 42-45.

[6] Achterberg, T. (2007). Constraint integer programming. zur erlangung des akademischen grades doktor der Naturwissenschen. Berlin, der Technischen Universitat Berlin, der Fakultät II - Mathematik und Naturwissenschaften.

[7] Applegate, D.; Bixby, R.; Chvátal, V. and Cook, W. (1995). Finding cuts in the TSP. Technical Report 95-05, DIMACS, Rutgers University.

[8] Applegate, D.; Bixby, R.; Chvátal, V. and Cook, W. (1998). On the solution of TSP. Documenta Mathematica Journal der Deutschen Mathematiker – Vereinigung, International Congress of Mathematicians, pp. 645 – 656.

[9] Eckstein, J.; Phillips, C.A. and Hart,W.E.; PICO. (2001). An object-oriented framework for parallel branch-and-bound, in Proc. Inherently Parallel Algorithms in feasibility and Optimization and Their Applications, pp. 219-265.

[10] Linderoth, J.T. and Savelsbergh, W.P. (1999). A computational study of search strategies in mixed integer programming, INFORMS Journal on Computing, 11, pp. 173-187.

[11] Andrew G. and Sandholm T. (2011) Information-theoretic approaches to branching in search, Discrete Optimization, 8, pp. 147–159.

[12] Glankwamdee, W and Linderoth, J. (2006). Lookahead Branching for Mixed Integer Programming. Technical Report 06T-004. Lehigh University. Department of Industrial and Systems Engineering.

[13] Derpich I. and Macuada C. (2014) A branching variables rule for the B & B algorithm based on the flatness of the polyhedron Advances in Engineering Mechanics and Materials, ISBN:978-1-61804-241-5, Eurompent Review, pp 226-231.

[14] Beasly, J. E. (1990) Collection of test data sets for a variety of OR problems on line. Available at: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html.