G. Muratore

A NEW ENUMERATION SCHEME TO SOLVE DISCRETE OPTIMIZATION PROBLEMS WITH GRID COMPUTING

R. 645 Settembre 2006

Gabriella Muratore – Dipartimento di Matematica ed Informatica, Università di Catania, viale A.Doria n 6 - 95125 Catania, Italy. Email: muratore@dmi.unict.it
Email: gabriellamuratore@yahoo.com.

This work was done while the author was visiting IASI-CNR

ISSN: 1128–3378

Abstract

In this paper we present an efficient method for generating, in order with respect to a cost vector, all binary vectors of a given dimension n. The method can be useful when the feasible region is described by non smooth (non linear, non differentiable, etc.) functions or when a functional representation of the feasible set does not even exists and only feasibility tests can be carried out. We also present a way of “slicing” the feasible region and use a grid computing approach to solve hard problems.

Key words: Non-Linear Problems; Black Box Problems; Grid Computing.

1. Introduction

Most of the work in constrained optimization is about *well behaved* functions where *well behaved* encompasses some assumption of continuity, differentiability, convexity, etc. There are situations, though, (mainly coming from real life applications) where it is not possible to express all the constraints through appropriate smooth functions or where the relationship among all the variables involved is not clearly defined. Also, it is customary to look for "one" optimal solution, while, sometimes, would be better having a set of optimal or near optimal solutions.

Just to give a concrete example, suppose that a pharmaceutical company is testing n active principles for curing a given disease. It wants to try k substances at the time and each of these combinations is called a remedy. Each substance i has, without interaction with the others, a known noxious level c_i, so that, a remedy x has a noxious level at least equal to the sum of its components noxious levels. It is unknown, yet, if the interaction among the substances in a given remedy will increase the overall remedy noxious level (more testing is necessary) but experts agree that is very unlikely that would decrease it. The biochemical team wants to find the p remedies with the lowest noxious level and that, when tested in vitro for few seconds, achieve some bio-targets. More complex lab-test (and future investments) will then concentrate on the p remedies. Here, trying to describe the "feasible" set through mathematical functions is basically hopeless: we are better off generating, in order with respect to the noxious level, candidate remedies and test them for "feasibility" (i.e.) carrying out our bio-tests on them.

There are many other optimization problems that could be solved by this simple generation approach or, better, variations of it.

For example, let's consider the following optimization problem:

$$\begin{aligned} \text{Min} \quad cz & \\ s.t. & \\ Az = b & \\ f_i(z,y)z_i = 0 & \qquad \forall\ i = \{1,\dots n\} \\ z_i \in \{0,1\} & \qquad \forall\ i = \{1,\dots n\} \\ y \in R^m & \end{aligned}$$

where $f_i()$ are any function (non linear, non differentiable, non convex, etc.) which is "active" if and only if the corresponding variable z_i is set to 1. This is, in its generality, a "difficult" optimization problem and, if n is of moderate size, a search scheme, based on carefully generating candidate solutions, could give better results than other methods for any value of m. More, we may have indexed and priced, through a combination of *0-1*, a collection of set, and we may ask for the lower price non-empty set.

As another example, let us consider a chance constrained program (i.e.) a stochastic program of the form:

$$\begin{aligned} \text{Min} \quad cx & \\ s.t. & \\ Ax \geq b & \\ x_i \in \{0,1\} & \qquad \forall\ i = \{1,\dots n\} \end{aligned}$$

where A is a stochastic matrix with a know probability distribution for each of its entry.[1] We can, as usual, solve its deterministic equivalent, if it is tractable, but, again, a simple search scheme may give a solution faster,if we are able to compute the convolution function of each row. The clear advantage is that the dimension of the problem does not increase as it is usually the case by using the deterministic equivalent formulation. Hence, this scheme can be handy if the number of the x variables is relatively smaller than the sample space dimension.

From these examples, it follows that, in general, we would like to solve the following problem: given a real vector c and a set $E \subset B^n$ with the following property: "it is exists an *easy* procedure to verify if a given x belongs to E or not", find $F \subset E : |F| = p$ such that $\forall\, x \in E - F$ and $\forall\, y \in F$ is $cx \geq cy$.

In the sequel we illustrate a method for generating all the n-dimensional binary vectors in order with respect to the objective function cx, (i.e.) if x^i is generated before x^j then $cx^i \leq cx^j$. We can use this generation algorithm to solve our general optimization problem since an optimal solution is obtained as soon as a *feasible* vector is generated. Hence, at each generation step, a *feasibility* test needs to be carried out. After generating p feasible vectors we stop. Sometimes, it is possible to derive a lower LB and (or) an upper bound UB to the optimal value or exploit the information from some "tractable" constraint. In these cases, it is possible to narrow the search region to a subsequence $(y^j) \subset (x^i)$ of length l, (where $y^1 = x^p$ for some p) with the following property: x^u is *unfeasible* for all $u < p$ and x^v is *not optimal* for all $v > p + l - 1$. Again, we can generate all the elements in the subsequence in order with respect to the objective function so that we are optimal as soon as we are feasible, or, we may speed up the search by exploiting somehow the "tractable" constraints.

In the next two sections we will describe the generation algorithm. In the third we will describe how we can improve our search algorithm and implement a grid-based computational strategy to solve hard problems. In the last section we test valid inequalities for *0-1* programs.

2. The Generation Algorithm

In this section we will show how we can generate all n-dimensional binary vectors in order with respect to any given n-dimensional cost vector c, (i.e) x^i is generated before x^j if and only if $cx \leq cy$. Without loss of generality, we will assume throughout the paper that the vector c is ordered in non decreasing way (i.e) $c_i \leq c_j$ if $i < j$.

Let us indicate the set of all n-dimensional binary vectors by B^n. We have that $B^n = \bigcup_{k=0}^{n} B^n(k)$ where $B^n(k)$ are the binary vectors with exactly k positive components, $B^n(k) = \{x \in B^n / \sum_{i=1}^{n} x_i = k\}$. Let us see, first, how we can generate all the elements in $B^n(k)$ in order with respect to c.

Given a n-dimensional vector, x, with exactly k positive components, we can build a k-tuple, $(j_1, \ldots j_k)$, of distinct integer numbers in the set $N = \{1 \ldots n\}$ by just considering the position of the positive components in x, and, viceversa, given a k-tuple of distinct integers numbers belonging to N we can build a n-dimensional binary vector with exactly k positive components by setting $x_j = 1$ if j is an element of the k-tuple and $x_j = 0$ otherwise. For example, let $n = 10$ and $k = 6$. Given the vector $x = (0, 0, 1, 1, 0, 1, 1, 0, 1, 1) \in B^{10}(6)$ its corresponding *6*-tuple is $u = (3, 4, 6, 7, 9, 10)$. If we consider the k-tuple ordered in increasing way we have a bijection ϕ between the set $B^n(k)$ and the set $\Omega^k(n) = \{(i_1, i_2, \cdots, i_k) / i_j \in \{1 \ldots n\},\ i_j < i_l \text{ if } j < l\}$. Let us define the following pre-order relationships:

[1]if a_{ij} is a number then its probability is 1

Definition 2.1. *Let be $c \in \Re^n$. $\forall x, y \in B^n(k)$, $x \tilde{\leq} y$ iff $cx \leq cy$. Similarly $\forall x, y \in \Omega^k(n)$ $x \tilde{\prec} y$ iff $\sum_{i \in x} c_i \leq \sum_{i \in y} c_i$.*

It is obvious that ϕ is an isomorphism with respect to these orderings (i.e.) $\forall x, y \in B^n(k)$ $x \tilde{\leq} y$ iff $\phi(x) \tilde{\prec} \phi(y)$. Hence, given c, it is enough to generate all the elements in $\Omega^k(n)$ in order with respect to $\tilde{\prec}$.

Let us consider, now, a very special cost vector, $\hat{c}$ where $\hat{c}_i = i$ $\forall i = \{1, \dots n\}$. In this case, $\forall x, y \in \Omega^k(n)$ $x \tilde{\prec} y$ iff $\sum_{i=1}^k x_i \leq \sum_{i=1}^k y_i$. It is easy to see that the minimum vector in $\Omega^k(n)$ (with respect to $\hat{c}$) is $x^1 = (1, 2, \dots k)$ with $\sum_{i=1}^k x_i^1 = \frac{k*(k+1)}{2}$ and the maximum vector is $x^t = (n-k+1, n-k+2, \dots n)$ where $\sum_{i=1}^k x_i^t = k*(n-k) + \frac{k*(k+1)}{2}$. We can then partition $\Omega^k(n) = \bigcup_{m=0}^{k(n-k)} T_m^k$ where $T_m^k = \{x^i \in \Omega^k(n) / \sum_{j=1}^k x_j^i = m + \frac{k(k+1)}{2}\}$. Each $T_m^k \neq \emptyset$ since, if $x^u \in \Omega^k(n)$, $u \neq t$ and $\sum_{i=1}^k x^u = H$ we can generate a vector $x^v \in \Omega^k(n)$ with $\sum_{i=1}^k x_i^v = H+1$ by appropriately increasing exactly one component of x^u. By construction, $x \tilde{\prec} y$ (with respect to $\hat{c}$) if $x \in T_m^k$ and $y \in T_{m+j}^k$ for some $j \geq 0$. Hence, we can generate all the elements in $\Omega^k(n)$ in order with respect to $\hat{c}$ by generating $T_0^k, T_1^k, T_2^k, \dots, T_t^k$. Before proceeding further let us give an example. Let $n = 7$ and $k = 4$. We have $\hat{c} = (1, 2, 3, 4, 5, 6, 7)$, $x^1 = (1, 2, 3, 4)$, $x^t = (4, 5, 6, 7)$. From $x^1 \in T_0^4$ we can generate the point $x^2 = (1, 2, 3, 5) \in T_1^4$; from $x^2 \in T_1^4$ we can generate two points in T_2^4, namely $x^3 = (1, 2, 3, 6)$ and $x^4 = (1, 2, 4, 5)$. Note that $\sum_{i=1}^4 x_i^3 = \sum_{i=1}^4 x_i^4 = 2 + \frac{4*5}{2} = 12$. In general, given the *k*-tuple $x = (i_1, i_2, \dots i_j \dots i_k)$ we consider the first index j such that $i_j \neq j$. Then, if $i_j + 1 < i_{j+1}$, we generate the *k*-tuple $y = (i_1, i_2, \dots i_{j-1}, i_j + 1, \dots i_n)$ and, if $i_{j-1} + 1 < i_j$, we generate the *k*-tuple $z = (i_1, i_2, \dots, i_{j-2}, i_{j-1} + 1, i_j, \dots, i_k)$. As an example, let's consider the *4*-tuple $x = (1, 2, 4, 6)$. The first index j such that $i_j \neq j$ is $j = 3$. Since by increasing $i_3 = 4$ by *1* we have $i_3 + 1 = 5 < i_{3+1} = 6$ we can generate the *k*-tuple $y = (1, 2, 5, 6)$, and, since $i_{j-1} + 1 = i_2 + 1 = 3 < i_3 = 4$ we can generate the *k*-tuple $z = (1, 3, 4, 6)$.

Let us consider, then, the following:

Definition 2.2. *Let's consider the relationship:*

$$G_k^n \colon \Omega^k(n) \to \Gamma(\Omega^k(n))$$

$$\begin{array}{lcccll}
(1, 2, \dots, k) & = & x^1 & \to & \{(1, 2, \dots, k-1, k+1)\} & \textit{if } k < n \\
 & & x^1 & \to & \emptyset & \textit{otherwise}
\end{array}$$

$$\begin{array}{lcccll}
(i_1, i_2, \dots i_k) & \neq & x^1 & \to & \{F_1^j(x), F_2^j(x)\} & \textit{where } j = min\{l \,/\, i_l \neq l\}
\end{array}$$

$$F_1^j \colon \Omega^k(n) \to \Omega^k(n) \cup \{\emptyset\}$$

$$\begin{array}{lcll}
(i_1, i_2, \dots \mathbf{i_j}, \dots i_k) & \to & (i_1, i_2 \dots i_{j-1}, \mathbf{i_j + 1}, i_{j+1} \dots, i_k) & \textit{if } i_j + 1 < i_{j+1} \textit{ when } j < k \\
 & & & \textit{if } i_j + 1 \leq n \textit{ when } j = k \\
(i_1, i_2 \dots \mathbf{i_j} \dots i_k) & \to & \emptyset & \textit{otherwise}
\end{array}$$

$$F_2^j \colon \Omega^k(n) \to \Omega^k(n) \cup \{\emptyset\}$$

$$\begin{array}{lcll}
(i_1, i_2 \dots \mathbf{i_j} \dots i_k) & \to & (i_1, i_2 \dots \mathbf{i_{j-1} + 1}, i_j \dots i_k) & \textit{if } i_{j-1} + 1 < i_j \textit{ when} j > 1 \\
(i_1, i_2 \dots \mathbf{i_j} \dots i_k) & \to & \emptyset & \textit{otherwise}
\end{array}$$

Definition 2.3. [2] *$G^n_k(A) = \bigcup_i \{Y_i / Y_i \in \Gamma(\Omega^k(n))$ and $\exists\, x \in A \,/\, G^n_k(x) = Y_i\}$ where $A \subseteq \Omega^k(n)$.*

We can then prove:

Lemma 2.1. *$G^n_k(T^k_m) = T^k_{m+1}$ for all k: $1 \leq k < n$ and m such that $0 \leq m < k(n-k)$. When $k = n$ is $\Omega^n(n) = \{(1, 2, \dots, n)\} = T^n_0$ and $G^n_n(T^n_0) = \emptyset$.*

Proof. The inclusion $G^n_k(T^k_m) \subseteq T^k_{m+1}$ is obvious. Let us show $G^n_k(T^k_m) \supseteq T^k_{m+1}$. We need to prove that $\forall\, x \in T^k_{m+1} \,\exists\, y \in T^k_m : G^n_k(y) = x$. Let $g = min\{l/x_l \neq l\}$. If $x_g = g + 1$ then $F^{g+1}_2(y) = x$ where $y = (1, 2, \dots, g-1, g, x_{g+1}, \dots x_k) \in T^k_m$; if $x_g > g + 1$ then $F^g_1(y) = x$ where $y = (1, 2, \dots, g-1, x_g - 1, x_{g+1}, \dots, x_k) \in T^k_m$. Either cases $T^k_{m+1} \subseteq G^n_k(T^k_m)$. ∎

Lemma 2.2. *The relationship G^n_k is "strongly injective" (i.e.) $\forall\, x, y \in \Omega^k(n)$, $x \neq y$ we have that $V \cap W = \emptyset$ where $V = G^n_k(x), W = G^n_k(y)$, and $V, W \in \Gamma(\Omega^k(n))$.*

and hence:

Corollary 2.1. *Starting from the tuple $(1, 2, \dots, k-1, k) \equiv T^k_0$ and applying repeatedly G^n_k, we can generate all elements in $\Omega^k(n)$ in order with respect to $\hat{c}$ (i.e.) $\bigcup_{p=1}^{k(n-k)} (G^n_k(1, 2 \dots k-1, k))^p = \Omega^k(n) - \{(1 \dots k)\}$. The complexity of this generation procedure is $O(\begin{pmatrix} n \\ k \end{pmatrix} k)$, where $O(k)$ is the time to actually write each k-tuple and $|\Omega^k(n)| = \begin{pmatrix} n \\ k \end{pmatrix}$.*

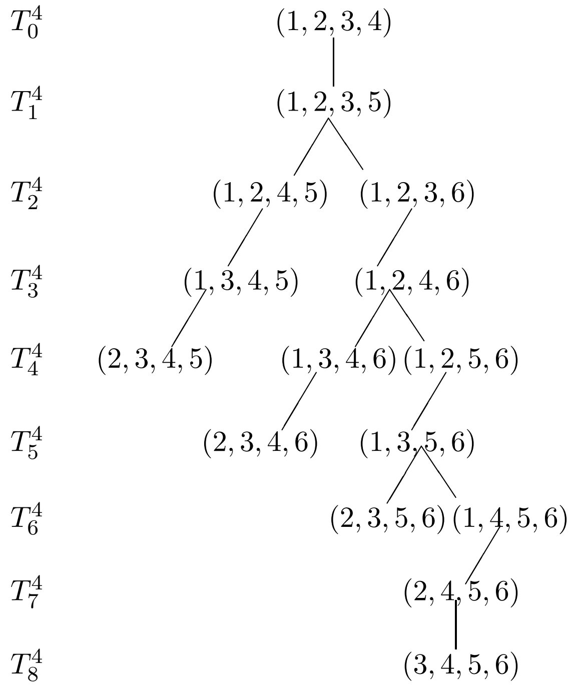
Note 2.1. *We can graphically represents $\Omega^k(n)$ by a binary tree of height $k(n-k)$ rooted at $T^k_0 = (1, 2, \dots, k)$ in which the two sons of each element are computed by G^n_k. Through this paper we will be assuming that the right son of x is $F_1(x)$ and that the left son is $F_2(x)$. Each "generation" or level of the tree coincides with T^k_m for some m. Note that $\forall\, m$, $0 \leq m < k(n-k)$, $\forall\, x \in T^k_m$ we have that $[G_k(x)]^i$ are contiguous elements at level $m + i$. Through this paper we will refer to this tree as a $G_k(n)$ tree.*

In figure 1 we have represented the set $\Omega^4(6)$ by its $G_4(6)$ tree.

Note 2.2. *Consider $\forall\, x \in \Omega^k - \{(1, 2, \dots, k)\}$. Let $x = (x_1, x_2 \dots x_k) = (1+y_1, 2+y_2 \dots k+y_k)$ where $y_i \leq y_j$ for $i < j$, $0 \leq y_i \leq n-k$ for all $i = \{1, 2, \dots, k\}$ and y_i integer for all i. Let $p = min\{\, l \,/\, x_l \neq l\} = min\{\, l \,/\, y_l \neq 0\}$. If $x_p = p+1$ then $x = \Pi^{p+1}_{j=k-1} F^j_2 \circ (F^j_1)^{y_j - 1} \circ \dots \circ F^k_2 \circ (F^k_1)^{y_k}(1, 2, \dots, k)$; otherwise $x = (F^p_1)^{y_p - 1} \Pi^{p+1}_{j=k-1} F^j_2 \circ (F^j_1)^{y_j - 1} \circ \dots \circ F^k_2 \circ (F^k_1)^{y_k}(1, 2, \dots, k)$. The path to generate x by G_k is unique.*

So far we have a method to build an ordered sequence of *0-1* vectors with exactly k positive components only when the cost vector is a very special one (i.e.) $\hat{c}_i = i$ for all $i = \{1 \dots n\}$. What happens if we change the cost vector? Well, we no longer have that $x \in T^k_m$ and $y \in T^k_{m+j}$ $\Rightarrow$ $x \tilde{\prec} y$ but the following holds true:

[2] In order not to get notation heavy we do not always specify the empty set. The reader will easily figure out the case.



T_0^4 (1, 2, 3, 4)
T_1^4 (1, 2, 3, 5)
T_2^4 (1, 2, 4, 5) (1, 2, 3, 6)
T_3^4 (1, 3, 4, 5) (1, 2, 4, 6)
T_4^4 (2, 3, 4, 5) (1, 3, 4, 6) (1, 2, 5, 6)
T_5^4 (2, 3, 4, 6) (1, 3, 5, 6)
T_6^4 (2, 3, 5, 6) (1, 4, 5, 6)
T_7^4 (2, 4, 5, 6)
T_8^4 (3, 4, 5, 6)

Figure 1: The set $\Omega^4(6)$ via its $G_4(6)$ tree

Lemma 2.3. *Assume the vector c is ordered in non decreasing way. Then* $\forall\, x \in \Omega^k(n)$, $\forall\, p > 0$ *and* $\forall\, y \in [G_k^n(x)]^p$ *we have* $x \tilde{\prec} y$. *Also,* $x \tilde{\prec} y \;\; \forall y \in T_m^k \;\; \Rightarrow \;\; x \tilde{\prec} z \;\; \forall z \in T_{m+j}^k$, $\forall\, j \geq 1$.

Proof. *[Proof]*
The first part immediately follows since the k-tuple x and $G_k^n(x) = y$ *differ only by one component h, the* j^{th} *or* $(j-1)^{th}$, *and* $c_{x_h} \leq c_{y_h}$ *since* $y_h = x_h + 1$ *(by the very definition of* G_k^n*) and c is ordered in non decreasing way. As to the second part, it is enough to prove it for* $j = 1$. *Let x be the minimum over* T_m^k *and z the minimum over* T_{m+1}^k. *By Lemma (2.1) and (2.2)* $\exists\,!\; w \in T_m^k$ *such that* $G_k^n(w) = z$. *By the first part of this Lemma we have* $x \tilde{\prec} w \tilde{\prec} G_k^n(w) = z$. ∎
By this Lemma, we can generate all elements in $\Omega^k(n)$ in order with respect to any cost vector c via a simple iterative algorithm. Starting from $x^1 = (1, 2, \ldots, k)$ it is enough to apply, at each iteration, G_k^n to the minimum of all generated k-tuples. At each iteration, i, the minimum of all generated k-tuples will give x^i since k-tuples which have not been generated yet are greater or equal than some k-tuple generated at iteration $j < i$. In table (1), L^i is the set of k-tuples generated at step i by this algorithm applied to the set $\Omega^4(6)$ when $c = (1, 3, 6, 6, 7, 9)$.

Note 2.3. *Given the set* $\Omega^k(n)$ *and a n-vector c, we can represent it through a weighted, binary tree rooted at* $(1, 2, \ldots, k)$, *where the tree is the* $G_k(n)$ *tree previously defined and the weight of each arc belongs to the set* $D = \{d_1, d_2, \ldots, d_{n-1}\}$ *where* $d_i = c_{i+1} - c_i$. *In particular, the weight of the arc that links a vector* $x = (1, 2, \ldots, j-1, x_j, \ldots, x_k)$ *to the vector* $F_1^j(x) = (1, 2, \ldots, j-1, x_j + 1, \ldots, x_k)$ *is* $d_{x_j} = c_{x_j+1} - c_{x_j}$ *and the weight of the arc that links a vector* $x = (1, 2, \ldots, j-1, x_j, \ldots, x_k)$ *to the vector* $F_2^j(x) = (1, 2, \ldots, j-2, j, x_j, \ldots, x_k)$ *is* $d_{j-1} =$

Table 1: The iterative algorithm applied to $\Omega^4(6)$ and $c = (1,3,6,6,7,9)$

Iteration i	List L^i	$x^i = min\{L^i\}$	$\sum_{j\in x^i} c_j$
1	{(1,2,3,4)}	(1,2,3,4)	16
2	{(1,2,3,5)}	(1,2,3,5)	17
3	{(1,2,3,6),(1,2,4,5)}	(1,2,4,5)	17
4	{(1,2,3,6),(1,3,4,5)}	(1,2,3,6)	19
5	{(1,3,4,5),(1,2,4,6)}	(1,2,4,6)	19
6	{(1,3,4,5),(1,2,5,6),(1,3,4,6)}	(1,3,4,5)	20
7	{(1,2,5,6),(1,3,4,6),(2,3,4,5)}	(1,2,5,6)	20
8	{(1,3,4,6),(2,3,4,5),(1,3,5,6)}	(1,3,4,6)	22
9	{(2,3,4,5),(1,3,5,6),(2,3,4,6)}	(2,3,4,5)	22
10	{(1,3,5,6),(2,3,4,6)}	(1,3,5,6)	23
11	{(2,3,4,6),(2,3,5,6),(1,4,5,6)}	(1,4,5,6)	23
12	{(2,3,4,6),(2,3,5,6),(2,4,5,6)}	(2,3,4,6)	24
13	{(2,3,5,6),(2,4,5,6)}	(2,3,5,6)	25
14	{(2,4,5,6)}	(2,4,5,6)	25
15	{(3,4,5,6)}	(3,4,5,6)	28

$c_j - c_{j-1}$. *By Note (2.2) we have: to any vector x such that $\sum_{i\in x} c_i = R$ corresponds a path in the tree from the root to x of weight $R - \sum_{j=1}^{k} c_j$ and viceversa.*

Now, it is an easy matter to generate, in order with respect to any cost vector c, all the n-dimensional binary vector in B^n.

3. The Order and Bound Algorithm

In many cases, it is possible to derive a lower bound LB and an upper bound UB to the optimal objective value. In these cases, it is possible to speed up the algorithm by limiting the search to a portion of the G tree. As before, we will first show how to generate the subsequence $(y^i) \subset (x^i) \subseteq B^n(k)$ and then extend the result to B^n. Let us consider the following IP programs[3] (or their relaxation):

$$\begin{array}{rlrl}
P_1: & \text{Min } \displaystyle\sum_{i=1}^{n} ix_i & P_2: & \text{Max } \sum_{i=1}^{n} ix_i \\
& s.t. & & s.t. \\
& cx \geq LB & & cx \leq UB \\
& x \in S & & x \in S \\
& x \in B^n(k) & & x \in B^n(k)
\end{array}$$

where S represent some *tractable* constraints[4], UB a known upper bound to the problem and LB a lower bound to the optimization problem restricted to $B^n(k)$.

[3]We assume, without loss of generality for rational data, that the cost vector is integer

[4]S can be empty

If V_1^k and V_2^k are, respectively, their optimal solution values we need to limit the search to the region

$$T^k = \bigcup_{m=m^*}^{m^{**}} T_m^k$$

where $m^* = \lceil V_1^k \rceil - \frac{k(k+1)}{2}$, $m^{**} = \lfloor V_2^k \rfloor - \frac{k(k+1)}{2}$ and T_m^k are as previously defined (i.e.)

$$T_m^k = \{x^i \in \Omega^k(n) / \sum_{j=1}^{k} x_j^i = m + \frac{k(k+1)}{2}\} \simeq$$

$$\phi^{-1}(T_m^k) = \{x \in B^n(k) : \sum_{i=1}^{n} ix_i = \hat{c}x = m + \frac{k(k+1)}{2}\}$$

Hence $(y^i) \subseteq \phi^{-1}(T^k)$ since all points in $\phi^{-1}(T_i^k)$ for all $i < m^*$ are not *feasible* (they have an objective value strictly less than the known lower bound LB or violate some constraint in S) and all points in $\phi^{-1}(T_i^k)$ for all $i > m^{**}$ are not *optimal* (they have an objective value strictly greater than the known upper bound UB or violate some constraints in S). In order to generate all points of (y^i) in order with respect to any cost vector c we may just slightly modify our iterative algorithm as follows: $y^1 = min\{x \in \phi^{-1}(T_{m^*}^k)\} = min\{x \in B^n(k) : \sum_{i=1}^n ix_i = \lceil V_1 \rceil\}$ where by *min* we mean with respect to the objective function cx (i.e.) y_1 is the optimal solution of optimizing cx over $\phi^{-1}(T_{m^*}^k)$. At each iteration i, y^i is the minimum over $\phi^{-1}(T_{m^*}^k)$ except those y^j already generated (i.e) $j < i$ or it corresponds, via the isomorphism ϕ, to the minimum of the points generated via the G_k^n function applied to previously generated $\phi(y_j)$. In formula: $y^i = min(A^i \cup B^i)$ where $A^i = \{z_l \in B^n(k) : z_l = \phi^{-1}(G_k^n(\phi(y^j)))\, for\ j < i; G_k^n(\phi(y^j)) \in T^k; z_l \neq y^j\ \forall j < i\}$ and $B^i = \{z_l \in B^n(k) : z_l = min\{\phi^{-1}(T_{m^*}^k) - \{y^j\ \forall j < i\}\}$. We can generate A^i starting at y^1 by using the same iterative algorithm showed in Table 1. Generating B^i entails solving {0-1} integer programs, which we know how to do quite well. For example, after finding $y^1 = min\{x : x \in \phi^{-1}(T^{m*})\}$ via, for example, a branch and bound algorithm, instead of stopping, we find the next best solution and so on. In the next section we will show that there are cases in which branching strategies based on the structural properties of T_m^k can greatly improve standard algorithms for solving IP programs over $\phi^{-1}(T_m^k)$ and hence for generating B^i.

In order to generate $(y^i) \subset (x^i) \subseteq B^n$ we may again solve the (relaxed) IP programs P_1 and P_2 where, though, the last constraint has been substituted by $x \in B^n$ and where the lower bound LB is a lower bound for the entire problem. If V_1 and V_2 are their respective optimal solution and $m^* = \lceil V_1 \rceil$, $m^{**} = \lfloor V_2 \rfloor$ we have that the search region is:

$$T = \bigcup_{m=m^*}^{m^{**}} T_m$$

and T_m as defined in Corollary (4.1). Note also that $T \supseteq \bigcup_{i=a}^{b} T_i$ where $a = min_{\{k\}} \lceil V_1^k \rceil$ and $b = max_{\{k\}} \lfloor V_2^k \rfloor$. The inclusion may be strict.

Corollary 3.1. *For any m:* $1 \leq m \leq \frac{n(n+1)}{2}$ *the inequalities*

$$\sum_{i=1}^{n} x_i \geq k_1 + 1$$

$$\sum_{i=1}^{n} x_i \leq k_2 - 1$$

are valid for $\phi^{-1}(T_m) = \{x \in B^n : \sum_{i=1}^{n} ix_i = m\}$ *if* $\exists\, k_1$ *such that* $k_1 = max\{k : m > nk - \frac{(k-1)k}{2};\ 1 \leq k < n\}$ *and* $\exists\, k_2$ *such that* $k_2 = min\{k : m < \frac{k(k+1)}{2};\ 1 < k \leq n\}$.

Proof. We know that $\Omega^k(n) = \bigcup_{m=0}^{k(n-k)} T_m^k$ and that $T_m^k \subset T_{m+k(k+1)/2}$. Hence $\Omega^k(n) \subset \bigcup_{m=k(k+1)/2}^{nk-(k-1)k/2} T_m$. Since $f(k) = nk - (k-1)k/2$ and $g(k) = k(k+1)/2$ are strictly increasing we have that, for any $k \leq k_1$, $\Omega^k(n) \cap T_m = \emptyset$ and, for any $k \geq k_2$, $\Omega^k(n) \cap T_m = \emptyset$ and the result follows. ∎

We can generate $(y^i) \subset (x^i) \subseteq B^n$ in order respect to any cost vector c via the same iterative algorithm used for generating $(y^i) \subset B^n(k)$ by substituting G_k^n with G^n and $T_{m^*}^k$ with T_{m^*}. We may use the valid inequalities described in Corollary (3.1) when generating B^i.

4. Divide et Impera

Sometime, if we can exploit tractable constraints, its better to optimize "slice" by "slice" (i.e.) over each T_m^k (for appropriate values of k and m). We have seen in the previous section how to compute the m values, given k. The next Lemma will help in find values for k.

Lemma 4.1. *Let be X a convex set. Define $\theta(k)$ =Min $\{cx/1x = k, x \in X\}$ where $k \in \Re$. Then theta(k) is convex.*

Proof. Let us consider the functions $\gamma(k)$ =*Min* $\{cx/1x \leq k, x \in X\}$, $k \in \Re$ and $\phi(k)$ =*Min* $\{cx/1x \geq k, x \in X\}$, $k \in \Re$. It is a know fact (see, for example, [[3]]) that $\gamma(k)$ and $\phi(k)$ are convex functions. Moreover, $\gamma(k)$ is non increasing and $\phi(k)$ is non decreasing. Hence, $\exists !$ k^* such that $\gamma(k)$ is strictly decreasing for $k < k^*$ (if defined) and constant afterwards (i.e.) $\gamma(k) = \gamma(k^*)$. By the same argument, $\exists !$ k^{**} such that $\phi(k)$ is strictly increasing (if defined) for $k > k^{**}$ and constant before (i.e.) $\phi(k) = \phi(k^{**})$ for $k < k^{**}$. Since $\theta(k) = \gamma(k)\ \forall\, k \leq k^*$ and $\theta(k) = \phi(k)\ \forall\, k \geq k^{**}$ must be $k^{**} \geq k^*$ since, otherwise, $\theta(k^{**}) = \gamma(k^{**}) > \gamma(k^*) = \theta(k^*) = \phi(k^*) > \phi(k^{**}) = \theta(k^{**})$. Also, $\gamma(k) = \phi(k)\ \forall\, k^* \leq k \leq k^{**}$ and, hence, $\theta(k^*) = \theta(k^{**})$. In order to prove that $\theta(k) = \theta(k^*)$ for all k such that $k^* \leq k \leq k^{**}$ consider the following LPs:

$$\begin{array}{rlcrl} Q_1: & \text{Min}\ \ 1x & \qquad & Q_2: & \text{Max}\ \ 1x \\ & \quad s.t. & & & s.t. \\ & cx = \theta(k^*) & & & cx = \theta(k^*) \\ & \quad x \in X & & & x \in X \end{array}$$

We know that problem Q_1 has solution k^* and problem Q_2 has solution k^{**}. Hence, $\forall\, k : k^* \leq k \leq k^{**}$ it must exists $y \in X$ such that $cy = \theta(k^*)$ and $1y = k$. Hence $\theta(k) = \theta(k^*)$ for all such k. We have that $\theta(k)$ is strictly decreasing for $k < k^*$, constant for $k : k^* \leq k \leq k^{**}$ and strictly increasing for $k > k^{**}$ (i.e) is convex. ∎

Next, we will describe some structural properties of the sets T_m^k which can be used to devise optimal branching strategies to optimize over $\phi^{-1}(T_m^k)$ and its subsets. We will compare one of these branching strategies with those used by *Cplex*[5] on the same problem. Results are quite interesting and we may think of implementing a quite simple, grid-based, computational strategy to solve IP hard problems. Before we do that, let us study T_m^k.

Lemma 4.2. *For any* $1 \le k \le n$ *and* $\forall\, m:\ 0 \le m \le k(n-k)$, $\forall\ x = (x_1, x_2, \dots, x_k) \in T_m^k$

$$\lceil \frac{m}{k} \rceil + k \le x_k \le min\{m, n-k\} + k$$

Proof. This is easily understood by noting that $x = (x_1, x_2, \dots, x_k) = (1+y_1, 2+y_2, \dots, k+y_k)$, where $y_i \le y_j$ if $i < j$, y_i integer and $0 \le y_i \le n-k$, $\forall i = \{1, \dots, k\}$, and $x \in T_m(k) \Longrightarrow \sum_{i=1}^k x_i = m + \frac{k(k+1)}{2}$ (i.e.) $\sum_{i=1}^k y_i = m$. If we assume that $x_k < \lceil \frac{m}{k} \rceil + k$ we have that $y_k < \lceil \frac{m}{k} \rceil$ and hence $\sum_{i=1}^k y_i \le k * y_k < k * \frac{m}{k} = m$. Contradiction. It is also obvious that $y_k \le min\{n-k, m\}$ (i.e) $x_k \le min\{m, n-k\} + k$. ∎

Corollary 4.1. *The inequality*

$$\sum_{i=\lceil \frac{m}{k} \rceil + k}^{p+k} x_i \ge 1$$

where $p = min\{m, n-k\}$ *is valid for* $\phi^{-1}(T_m^k)$ $\forall\, k:\ 1 \le k \le n$ *and* $\forall\, m:\ 0 \le m \le k(n-k)$. *Also, if* $n-k > m$ *then* $x_i = 0$ *is valid for all* $i = \{m+k+1, \dots, n\}$, *and, if* $d = m-(k-1)(n-k) > 0$ *then* $x_i = 0$ *for all* $i = \{1, \dots, d\}$ *is valid for* $\phi^{-1}(T_m^k)$.

Proof. The first part immediately follows from Lemma(4.2). If $n-k > m$ then, for any $x \in T_m^k$, its maximum coordinate $x_k \le m+k$ and hence all points in $\phi^{-1}(T_m^k)$ necessarily have $x_i = 0$ for all $i:\ m+k+1 \le i \le n$. If $d = m-(k-1)(n-k) > 0$ it means that, for any $x \in T_m^k$, its minimum coordinate $x_1 \ge d$ and hence $x_i = 0$ for all $1 \le i \le d$ must be valid for $\phi^{-1}(T_m^k)$. ∎

Lemma 4.3. $T_m^k = \bigcup_{i=a}^b Q[i]$ *where* $a = \lceil \frac{m}{k} \rceil + k$, $b = min\{m, n-k\} + k$ *and* $Q[i] = \{(x_1, \dots, x_k) \in T_m^k :\ x_k = i,\ (x_1, \dots, x_{k-1}) \in T_s^{k-1}$ *and* $x_{k-1} < i\}$ *where* $s = m-(i-k)$.

If we assume the tree representation described in Note (2.1), the sets $Q[i]$, for $i = \{p+k, \dots \lceil \frac{m}{k} \rceil + k\}$, $p = min\{n-k, m\}$, are consecutive sets, from right to left, at level m in the $G_k(n)$ tree. By the recursive relationship in Lemma(4.3), we can generate T_m^k from right to left and any of its subsets $Q[i_{j+1}, \dots, i_{k-1}, i_k] = \{x \in T_m^k \,/\, x_{j+1} = i_{j+1}, \dots x_k = i_k\}$.

Now, suppose that we want to solve the following IP:

$$\begin{aligned} Min\ \ cx \\ s.t. \\ \sum_{i=1}^n i x_i = V \\ x \in S \\ x \in B^n(k) \end{aligned}$$

where S is a set (possibly empty) of linear constraints. We can use, as it is customary, a branch and bound/cut method.When we are at the root, though, instead of branching on any variable

[5]One of the best commercial IP solver

from 1 to n, we branch on the highest coordinate, that, by Lemma (4.2), can assume only the values i with $\lceil \frac{m}{k} \rceil + k \leq i \leq p + k$, $p = min\{(n-k), m\}$. We may think of branching on all the $q = p - \lceil \frac{m}{k} \rceil$ variables or to devise a *smart* partitioning strategy. For example, $A_1 = B_1 \cup B_2 \cup B_3$ where $A_1 = \{x_k \,/\, \lceil \frac{m}{k} \rceil + k \leq x_k \leq p + k\}$, $B_1 = \{x_k \,/\, \lceil \frac{m}{k} \rceil + k \leq x_k \leq v - 1\}$, $B_2 = \{x_k = v\}$ and $B_3 = \{v + 1 \leq x_k \leq p + k\}$. Once the highest coordinate has been fixed, we branch on the second highest and so on.

We have tested our branching strategy on one of the MIPLIB market share problem, namely *markshare1* which has 50 binary variables and 6 linear constraints. This type of problems are notoriously difficult to solve by standard branch and bound/cut, even for small size [1]. In order to have a pure *0-1* program (without converting integer variables into binaries) we consider the problem:

$$\begin{aligned} Min\ cx \\ s.t. \\ Ax \geq b \\ x \in B^n \end{aligned}$$

where $c_j = \sum_{i=1}^{m} a_{ij}$ for all $j = \{1, \ldots n\}$ and (A, b) are the linear constraints of *markshare1.* [6] In the rest of the paper, we will refer to this problem as *markshare1.* The specific partition we have used is quite simple: branching always on the highest coordinate. This choice is only due to the fact that we do not know if it is possible to implement in *Cplex* the more general branching strategy.

In table (2) we report our testing for various values of k and m, randomly chosen.

We have solved the same problem through *Cplex* (versions 8.0, 9.0 and 10.0) with automatic setting versus *Cplex* with a priority order on the variables (equivalent to our branching strategy). In table (3) we have synthesized results from table (2).

Same type of results holds for more data point. In table (4) we report KPI averages and other statistical information for $k = 23$ and $k = 24$ for 50 values of m.

Beside the fact, that on this small sample test, results are on the side of our branching strategy, we should remark that this is not necessarily the *"best"* strategy, since a *"smarter"* partition could give better results. Anyway, it is obvious that the *best* strategy is the one that merges together known branching selection methods (implemented in *Cplex*) with those based on the knowledge of the T_m^k structure. By merging together all these strategies, we believe there should be a significant improvement on KPI's on any class of problems. Finally, we have noted that, for example, we can solve *markshare1* on T_{600}^{25} by solving about 300 IP's (obtained by fixing variables between x_{50} and x_{41}), each of them being solved through our branching strategy in less than 2 minutes. Hence, if we could launch, at the same time, these 300 IP's we could solve *markshare1* on T_{600}^{25} in less than 2 minutes (assuming, as it is reasonable, that the time to launch each IP is negligible). In general, by considering at most 1024 IP's running at the same time, we could solve *markshare1* on any T_m^k in the order of few minutes. Also, with a known upper bound of 7302[7], we can limit our search region T between T_m^* and T_m^{**} where $m^* = 475$ and $m_{**} = 803$. Moreover, from Lemma(4.1), we have that the lower envelope of $\theta(k) = min_{\{k\}} P^k$ (where P^k is our optimization problem constrained on $B^n(k)$) is convex and hence we may limit

[6]Note that solving this problem is equivalent to solve the feasibility version of Market Share type of problems. Moreover, if its optimal solution is $q = \sum_{i=1}^{m} b_i$ or $q + 1$ then the optimal solution of the Market share is, respectively, 0 or 1.

[7]This value is easily obtained after running Cplex for few seconds

Table 2: Each problem is *markshare1* plus two constraints: $\sum_i x_i = k$ and $\sum_i ix_i = m{+}k(k{+}1)/2$ which constraint a solution in $\phi^{-1}(T_m^k)$

Solution Method	k	m+k(k+1)/2	Obj Value	Time sec	Iterations #	Nodes #
Standard Cplex 9.0	23	700	7292	7709.99	146,668,592	62,823,520
Special Branching 9.0	23	700	7292	7640.52	146,668,592	62,823,520
Standard Cplex 10.0	23	700	7292	5341.72	139,775,421	44,046,583
Special Branching 10.0	23	700	7292	6965.47	175,840,197	65,570,252
Standard Cplex 8.0	24	627	inf	37.52	850,798	232,798
Special Branching 8.0	24	627	inf	19.51	519,377	141,746
Standard Cplex 8.0	24	630	inf	333.74	7,722,656	2,005,304
Special Branching 8.0	24	630	inf	113.83	2,899,806	842,585
Standard Cplex 9.0	24	633	7295	700.83	16,454,185	4,038,088
Special Branching 9.0	24	633	7295	463.85	10,861,771	3,396,238
Standard Cplex 9.0	25	600	7296	57922.12	796,970,669	315,100,013
Special Branching 9.0	25	600	7296	9980.78	180,863,134	81,510,392
Standard Cplex 10.0	25	600	7296	19717.22	532,491,395	196,668,261
Special Branching 10.0	25	600	7296	8855.41	215,259,671	83,974,600
Standard Cplex 9.0	28	510	7302	32.13	910,699	348,499
Special Branching 9.0	28	510	7302	32.70	910,669	348,499

our search to few k value (in our case they are $k = 23, k = 24, k = 25$). Hence,we could solve, through a parallel approach, *markshare1* in a very reasonable time.

In general, the grid-computing approach that could be used to tackle IP hard problems is as follows:

1. Slice the feasible region through T_m^k (for appropriate values of k and m);

2. Launch at the same time, over an appropriate computing space (the grid), the IP associate to each T_m^k;

3. Each T_m^k IP problem will give birth (branching), within the grid, to autonomous IP's over an (appropriately chosen) partition of T_m^k;

4. Each computing node, when it solves its IP problem, will communicate the result to a single information point which, in the end, will contain the final result of the computation.

We may think of enhancing performance through a very simple coordination mechanism: each computing node in the grid (which is solving an IP) can access, at any time, a single information point where the best bound found, by any node at that time, is stored.

Table 3: A minus sign indicates that *Special Branching* decreases the performance indicator value (e.g., less time), with respect to *Standard Cplex*, a plus sign indicates otherwise.

Method Comp.	k	m+k(k+1)/2	Time Diff. %	Iteration Diff. %	Node Diff. %
Cplex vs. Branch.	23	700	-0.91%	0%	0 %
Cplex vs. Branch.	23	700	+30.40%	+25.80%	+48.87%
Cplex vs. Branch.	24	627	-92.31%	-63.81%	-64.24%
Cplex vs. Branch.	24	630	-193.19%	-166.32%	-137.99%
Cplex vs. Branch.	24	633	-51.09%	-51.49%	-18.90%
Cplex vs. Branch.	25	600	-480.34%	-340.65%	-286.58%
Cplex vs. Branch.	25	600	-122.66%	-147.37%	-134.20%
Cplex vs Branch.	28	510	+1.77%	0%	0%
Average			-113.54%	-92.98%	-74.13 %

Table 4: A minus sign indicates that *Special Branching* decreases the performance indicator value (e.g., less time) with respect to *Standard Cplex*, a plus sign indicates otherwise.

Method Comp.	k	Time Diff. %	Iteration Diff. %	Node Diff. %
Cplex vs. Branch.	23	-115.98%	-125.54%	-111.49 %
# of times KPI is negative		23/25	22/25	23/25
# of times KPI is zero		1/25	1/25	1/25
# of times KPI is positive		1/25	2/25	1/25
Max negative KPI		-18.18%	-2.52%	-25.89%
Min negative KPI		-293.70%	-320.66%	-261.48%
Average negative KPI		-126.54%	-144.10%	-121.63%
Max positive KPI		+10.93%	+25.35%	+10.13%
Min positive KPI		+10.93%	+6.36%	+10.13%
Average positive KPI		+10.93%	+15.86%	+10.13%
Cplex vs. Branch.	24	-101.06%	-86.27%	-68.50%
# of times KPI is negative		23/25	24/25	23/25
# of times KPI is zero		0/25	0/25	0/25
# of times KPI is positive		2/25	1/25	2/25
Max negative KPI		-7.84%	-15.00%	-4.86%
Min negative KPI		-206.04%	-206.66%	-199.20%
Average negative KPI		-112.88%	-94.32%	-80.49%
Max positive KPI		+66.67/5	46.02%	+74.86%
Min positive KPI		+2.99%	46.02%	+0.86%
Average positive KPI		+34.83%	46.02%	+37.86%
Average		-107.48%	-111.47%	-94.48 %

Acknowledgments

I wish to thank Paolo Ventura[8] for his fundamental help in coding the algorithms, for his uncountable tips on *Cplex* and overall support in this research. Many thanks also to Daniel Bienstock[9] and Giovanni Rinaldi[10] for allowing me to use their computing resources and for useful feedbacks on drafts.

[8]www.iasi.cnr.it
[9]www.corc.ieor.columbia.edu
[10]www.iasi.cnr.it

References

[1] K. Aardal, R. E. Bixby, C. A. J. Hurkens, A. K. Lenstra, J. W. Smeltink, " Market Split and Basis Reduction: Towards a Solution of the Cornuéjols-Dawande Instances", *Lecture Notes in Computer Science*, 1999.

[2] D. Avis, K. Fukuda, "Reverse Search for Enumeration", *Discrete Applied Mathematics*, 65, 21-46, 1996.

[3] M. S. Bazaraa, H. D. Sherali, C. M. Shetty, *Non Linear Programming- Theory and Algorithms*, Wiley, New York, 1993.

[4] U. U. Haus, M. Koppe, R. Weismantel, "The Integral Basis Method for Integer Programming", *Mathematical Methods of Operations Research*, 53, 353-361, 2001.

[5] H. W. Lenstra Jr.," Integer Programming with a Fixed Number of Variables", *Mathematics of Operations Research*, 8, 538-548, 1983.

[6] G.L. Nemhauser, L.A. Wolsey, *Integer and Combinatorial Optimization*, Wiley, New York, 1988.