RESEARCH PROJECT

The University Of Sheffield.

AMA 492- Implicit Enumeration Binary Integer Programming

Balas Algorithm

Registration No: 090248579

3 - June - 2010

1 Index

2 Introduction

We know that there are important economic problems which have mathematical models that are linear programs with integer variables. Among these problems are those that have variables taking only one of the values 0 or 1.

Such binary integer programs, serve as mathematical models for capital budgeting, project selection, pipeline or communications network design, structural design, switching circuit design, information retrieval, fault detection, design of experiments, facility location, truck dispatching, tanker routing, crew scheduling, machine sequencing, and a host of other decision problems involving logical alternatives.

Because of the importance of these decision problems the project considers a program developed in C++ that solves linear programs with variables constrained to take only one of the values 0 or 1 following the steps of the algorithm that the prestigious mathematician, Egon Balas, developed in 1965.

In this document we are going to study the basic ideas and outline of the algorithm, and subsequently we will analyze the algorithm in detail, showing interesting examples like the Diet Problem with 96 variables.

Finally we will show a tutorial of the application and we will draw conclusions of the operation of the algorithm.

3 Basic ideas and outline of the additive algorithm

The general form of a linear program with zero-one variables may be stated as follows:

$\min or \max\ Z = c^T x$

$Subject\ to$

$$Ax \leq \mathrm{b}$$

$$or$$

$$Ax \geq \mathrm{b}$$

$$x_j = 0\ or\ 1\ (j \in N)$$

Where

x= (x_j) is an n-component column-vector.

c= (c_j) is a given n-component row-vector.

A= (a_{ij}) is a given mxn matrix.

b= (b_j) is a given m-component column-vector.

4 Form of an Integer Binary Problem

Although we have defined the form of a linear program with zero-one variables problem, the Balas algorithm requires that the problem be put into a standard form, with the constraints inequalities of the form *less than*, and all the coefficients of the objective function (to be minimized) being nonnegative.

Any problem can be brought to this form by the following operations:

a) Replacing all equations by two inequalities.

b) Multiplying by -1 all inequalities of the form $\geq$

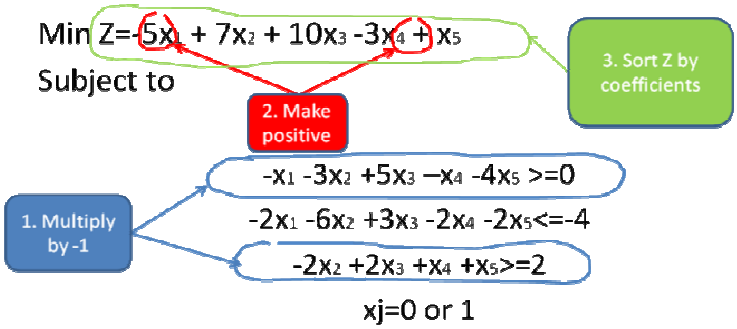
c) Setting xj as

xj for $cj \geq 0$ when minimizing; for $cj \leq 0$ when maximizing,

$(1 - xj)$ for $cj < 0$ when minimizing; for $cj > 0$ when maximizing.

So what we have to do is set all of the variables to zero to give the smallest value of Z. If we cannot do this without violating one or more constraints, then we prefer to set the variable that has the smallest index to 1. This is because the variables are ordered so that those earlier in the list increase Z by the smallest amount.

5 Form of an Integer Binary Problem – Example of transformation



5.1 Multiplying by -1 first and third constraints

Min Z= $-5x_1 + 7x_2 + 10x_3 - 3x_4 + x_5$

Subject to

We have the three constraints in less-equal form

$x_1 + 3x_2 - 5x_3 + x_4 + 4x_5 <= 0$

$-2x_1 - 6x_2 + 3x_3 - 2x_4 - 2x_5 <= -4$

$2x_2 - 2x_3 - x_4 - x_5 <= -2$

xj=0 or 1

5.2 Making x_1 and x_4 coefficients positive

Min Z= $5x_1 + 7x_2 + 10x_3 + 3x_4 + x_5$

Subject to

2.1 We have multiplied by -1 the negative coefficients

$(1-x_1) + 3x_2 - 5x_3 + (1-x_4) + 4x_5 <= 0$

$-2(1-x_1) - 6x_2 + 3x_3 - 2(1-x_4) - 2x_5 <= -4$

$2x_2 - 2x_3 - (1-x_4) - x_5 <= -2$

$x_j = 0$ or 1

2.2 We substitute in all the constraints x1 for (1-x1) and x4 for (1-x4)

Min Z= $5x_1 + 7x_2 + 10x_3 + 3x_4 + x_5$

Subject to

2.3 Making operations we get these three constraints

$-x_1 + 3x_2 - 5x_3 - x_4 + 4x_5 <= -2$

$x_1 - 6x_2 + 3x_3 + x_4 - 2x_5 <= 0$

$2x_2 - 2x_3 + x_4 - x_5 <= -1$

$x_j = 0$ or 1

5.3 Sorting Objective Function by Coefficients

Min Z= $x_5 + 3x_4 + 5x_1 + 7x_2 + 10x_3$

Subject to

3. If we sort the objective function by coefficients from minor to major we can find faster the best solution when we are covering the tree of solutions.

$-x_1 + 3x_2 - 5x_3 - x_4 + 4x_5 <= -2$

$x_1 - 6x_2 + 3x_3 + x_4 - 2x_5 <= 0$

$2x_2 - 2x_3 + x_4 - x_5 <= -1$

$x_j = 0$ or 1

6 Explanation of the algorithm for solving Integer Binary Problems

The main idea of the algorithm is to set all the n variables equal to 0 (minimizing), assign to certain variables the value 1 and after trying a part of all the two-possible combinations, one obtains either an optimal solution, or evidence of the fact that no feasible solution exists.

To explain how the algorithm works internally we are going to consider that we have the problem in the standard form

$min\ Z = c^T\ x$

$Subject\ to$

$$Ax \leq \mathrm{b}$$

$$x_j = 0\ or\ 1\ (j\ \in\ N)$$

Where

x= (x_j) is an n-component column-vector.

c= (c_j) is a given n-component row-vector.

A= (a_{ij}) is a given mxn matrix.

b= (b_j) is a given m-component column-vector.

6.1 Steps of the algorithm

6.1.1 Step 0:

- **Initialize:**

  Verify if the solution for the vector x=0 is feasible. If it is, we have finished, the vector 0 is the optimum.

- **Boundaries:**

  We are going to use two variables to indicate de upper and low boundaries for the values of the objective function Z.

  - Z_u is the upper bound and it is equal to the sum of the coefficients.
  - Z_l is the lower bound and it is the value of $c1$ and x=(1,0,0,…,0)^T.
  - We verify if x is feasible. If it is, this is the optimum.

6.1.2 Step 1:

Select one of the subsets of unverified solutions and we divided that subset into two, adding $x_j = 0$ and $x_j = 1$ when branching in the variable x_j (in the first iteration $j = 1$).

6.1.3 Step 2:

For each new subset, we fix x_{j+1} equal to 1 and the rest equal to 0, and we use this x to determine the value of the inferior boundary Z_l for the objective function Z in this subset.

6.1.4 Step 3:

We examine every subset of unverified solutions and we determine if it is just tested if:

- If $Z_l \geq Z_u$
- If it exists a constraint that can't be satisfied with any variable value assigned to the rest of the variables of this subset, that is, from $j + 1$ to n.
- If x is feasible. If it is satisfied we declare x as the solution and we assign to Z_u the value of Z_l

6.1.5 Step 4:

If we don't have more subsets to test, we stop. The last solution is the optimum. Else, we go to step 1.

7 Working example of the algorithm

We are going to use an example to show how the algorithm operates internally.

minimize $Z= 3x_1 + 5x_2 + 6x_3 + 9x_4 + 10x_5 + 10x_6$

Subject to:

$$(1)\quad 2x_1 - 6x_2 + 3x_3 - 4x_4 - x_5 + 2x_6 <= -2$$

$$(2)\quad 5x_1 + 3x_2 - x_3 - 3x_4 + 2x_5 - x_6 <= 2$$

$$(3)\quad -5x_1 + x_2 - 4x_3 + 2x_4 - 2x_5 + x_6 <= -3$$

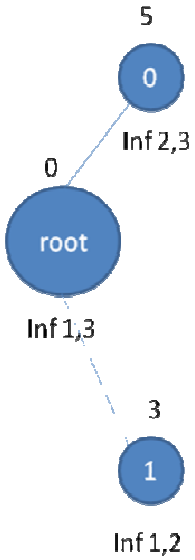
The terms in the objective function are written in increasing order, and that it is a minimization problem. So having $x_1 = 0$ and $x_2 = 1$ is worse than having $x_1 = 1$ and $x_2 = 0$ etc.

We start the tree in the Balas Algorithm with the root node. The solution at the root node is (0,0,0,0,0,0). The root node has an objective function value of 0, but this solution is infeasible. The first and third constraints are not satisfied.

Then we have to expand the tree. We will branch on x_1 because it has the lowest coefficient in the objective function.

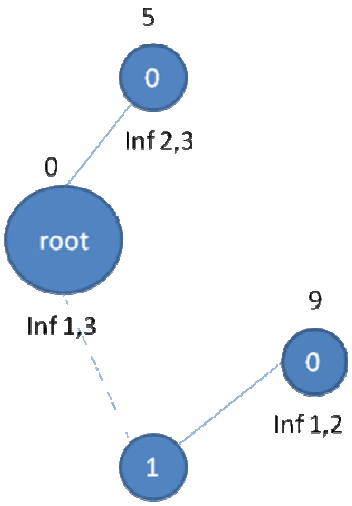
The objective function solution at node $x_1 = 0$ is (0,1,0,0,0,0). The value of the objective function is 5. But the solution is infeasible because it doesn't satisfy constraints 2 and 3.

For the node $x_1 = 1$ the objective function solution is (1,0,0,0,0,0). The value of the objective function is 3, but it is infeasible because it doesn't satisfy constraints 1 and 2.



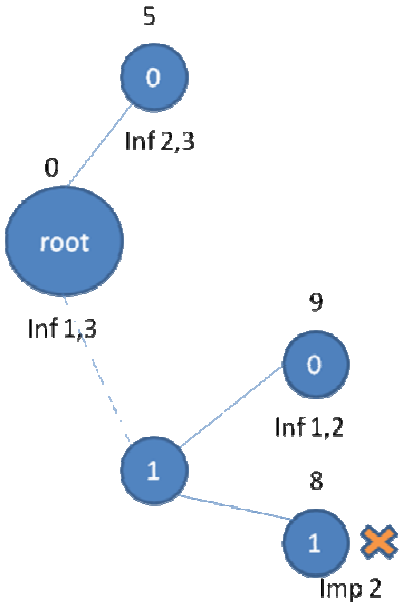
We now have two live nodes at the same level to expand. We will choose the node that has the smallest value of the bounding function.

We will now branch on x_2. First we look at setting x_2 to 0.



The bounding function solution at this node is (1,0,1,0,0,0). This gives a bounding function value of 9, but the solution is infeasible for the constraints 1 and 2.

Now let us set x_2 to 1. The bounding function solution at this node is (1,1,0,0,0,0).

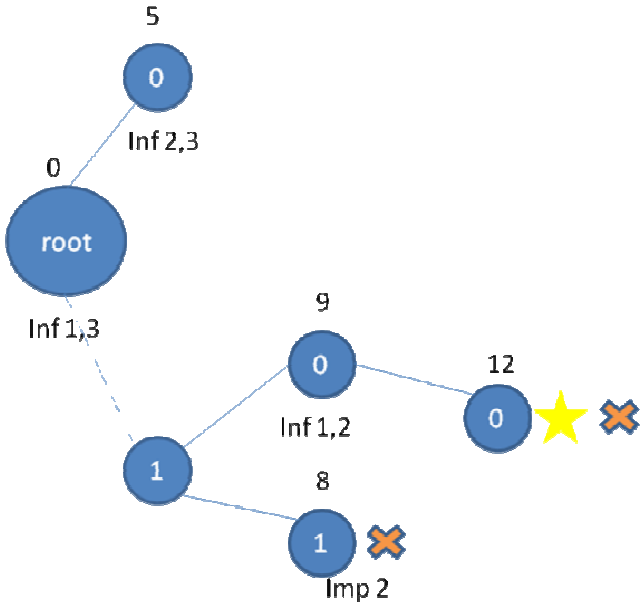


This gives a bounding function solution of 8. The solution is infeasible by constraint 2. If we try to decrease the left hand side making 1 the negative coefficients of x_3, x_4 and x_6 we will never get a value less than or equal to 2. Then we prune this branch.

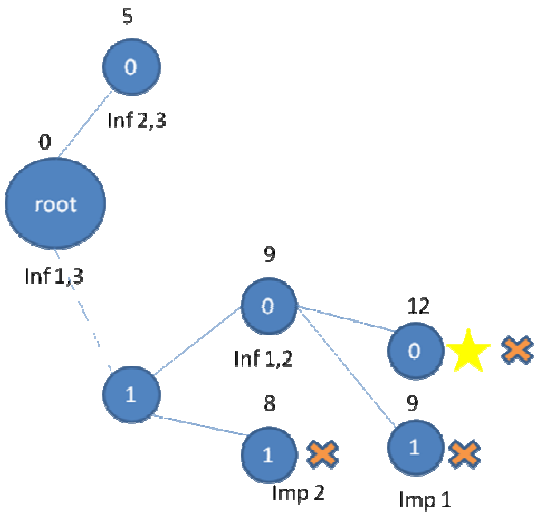
The next node that we will select for expansion is the node with the bounding function value of 9, because of the depth-first node selection strategy. Although there is a node with a smaller bounding function than 9, depth-first selection causes us to choose amongst the nodes just created.

We now branch on x_3. The bounding function solution for this node is (1,0,0,1,0,0). This gives a bounding function value of 12. This is our first feasible solution, so we will record it as our incumbent.

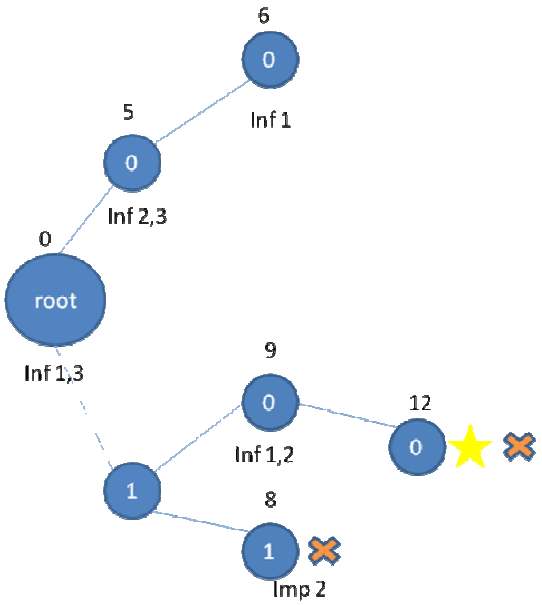
We also mark it as feasible. This node will not be expanded further because it is a minimization problem, and trying more non-zero variables will make the objective function larger.



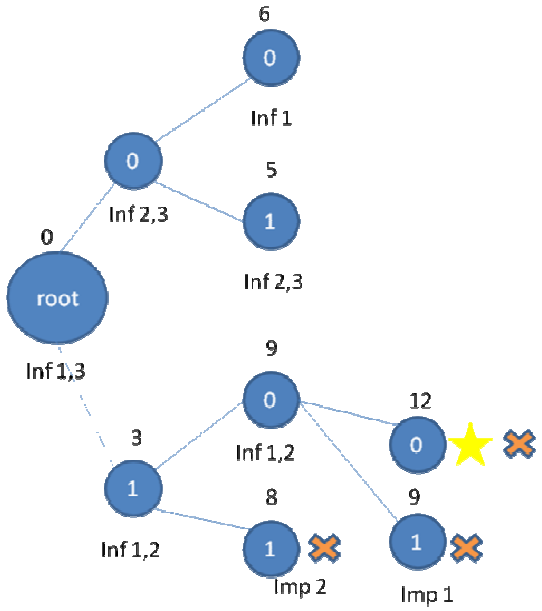
Let us look at the next node. The bounding function solution for this node is (1,0,1,0,0,0). The bounding function gives a value of 9. With this solution it is impossible to satisfy constraint 1. We will prune this node.



There are no more live nodes in this branch, so we will move to the node with the bounding function value of 5. We will branch on x_2. First let's look at setting x_2 to 0. The bounding solution at this node is (0,0,1,0,0,0) . This gives a bounding function value of 6. The solution is infeasible for constraint 1.

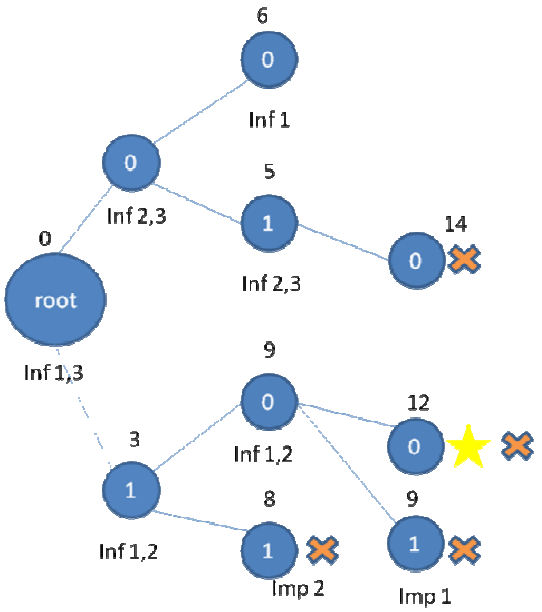


We will now look at the next node. The bounding function solution at this node is (0,1,0,0,0,0). This gives a bounding function solution of 5. The solution is infeasible for constraints 2 and 3.

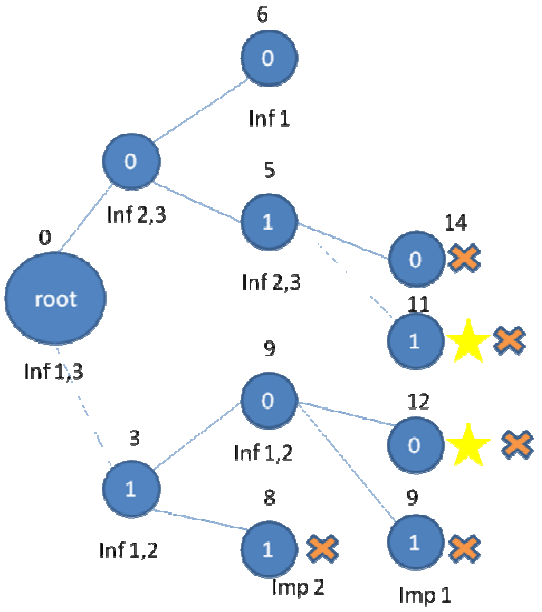


We are going to branch on the node with the smallest value for bounding function at this level in the depth-first search. We will be branching on x_3. First we will set x_3 to 0.

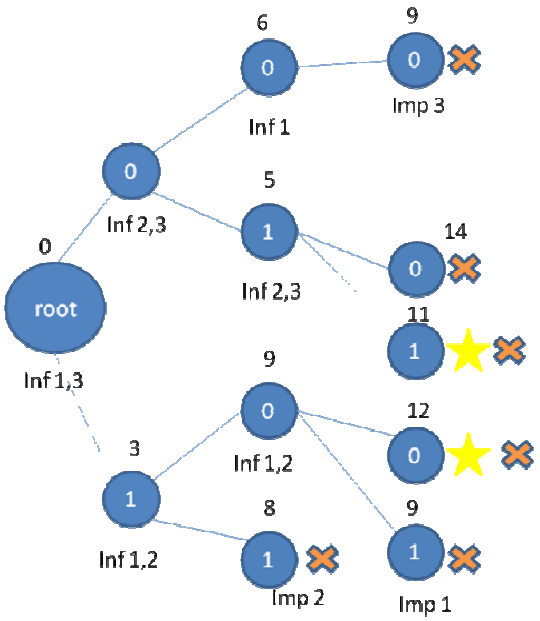
The bounding function solution for this node is (0,1,0,1,0,0). This gives a bounding function value of 14. The value of the bounding function is greater than the current incumbent (12). This branch cannot produce a solution that is better than the current incumbent, so this node is pruned.



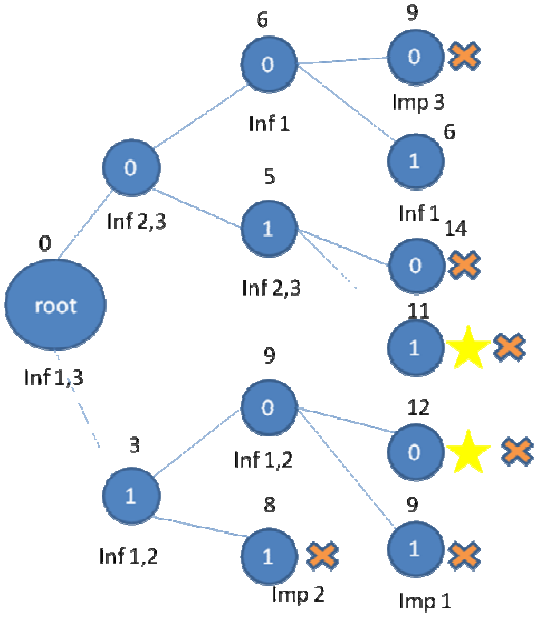
We now look the next node. The bounding function solution at this node is (0,1,1,0,0,0). This gives a bounding function value of 11. This value does not violate any constraint; therefore it is a feasible solution. This feasible solution is better than the current incumbent. This solution will take its place as the incumbent.



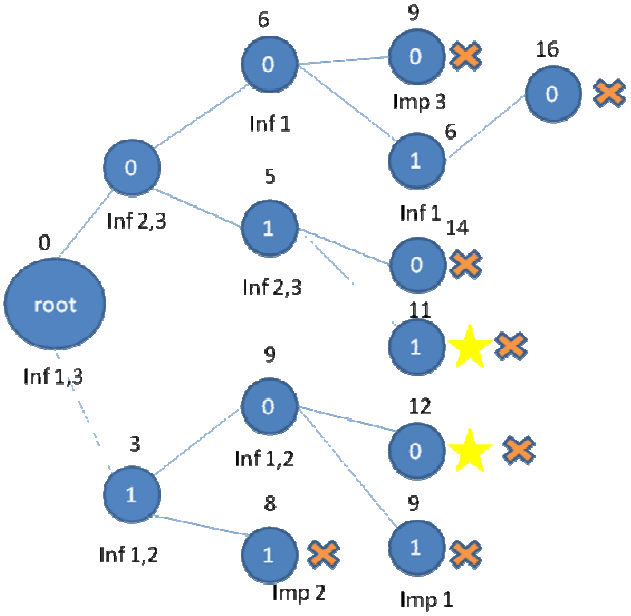
We now go back to the node with the bounding function value of 6. We will branch on x_3. The bounding solution for this node is (0,0,0,1,0,0). This gives a bounding function value of 9. This solution is impossible by constraint 3. The node is pruned.



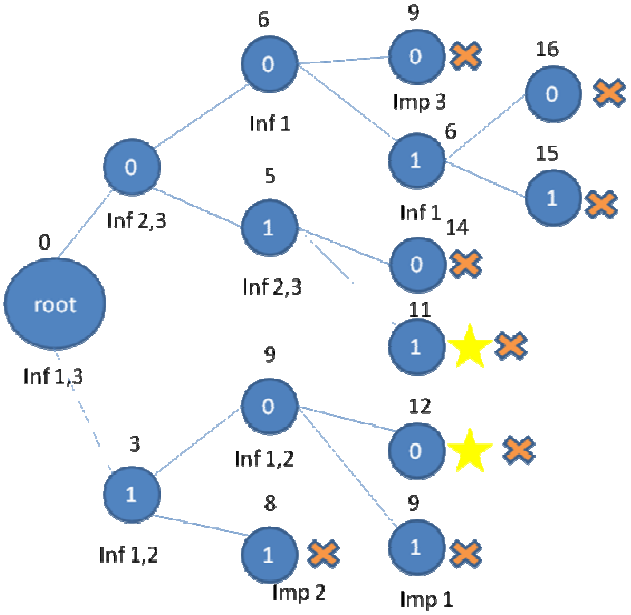
The bounding function solution for the next node is (0,0,1,0,0,0). This gives a bounding function value of 6. This solution is infeasible by constraint 1. This node is the only live node, so we will expand this node for x_4

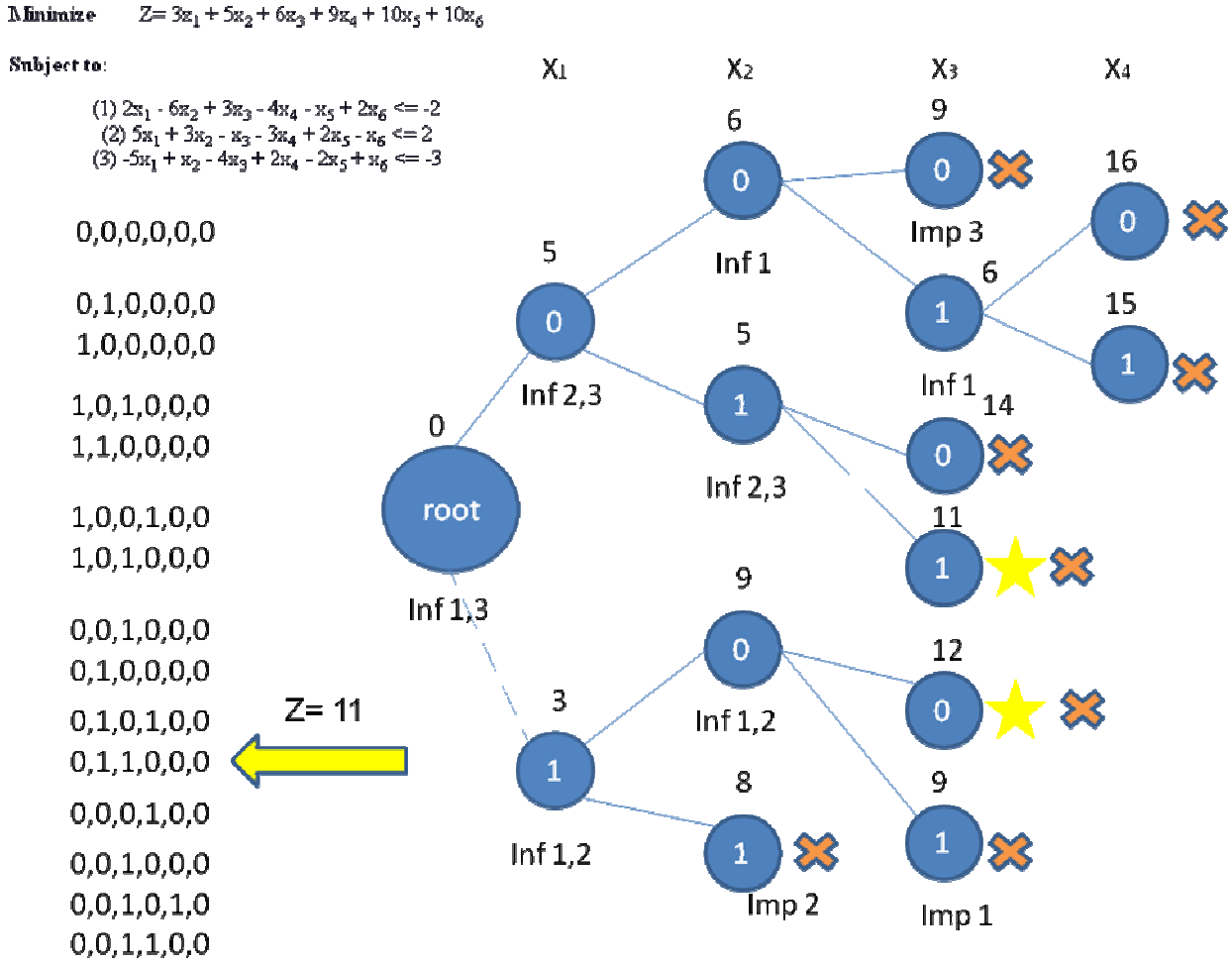


The bounding function solution for this node is (0,0,1,0,1,0). This node gives a bounding function value of 16. This value is greater than the incumbent. This node is pruned.



The bounding function solution at this node is (0,0,1,1,0,0). This gives a bounding function value of 15 that is greater than the incumbent. This node is pruned.





The optimum for this problem is Z=11.

8 Program

The program is a console executable that has been programmed in the object oriented language C++, with features such as data abstraction, encapsulation, modularity, polymorphism, and inheritance. The software that has been used is using the software Dev-C++ 4.9.9.4.

8.1 Source Code

The source code consists of two classes:

- *Queue.cpp*
- *BalasAlgorithm.cpp*

8.1.1 Queue.cpp

This class represents the *Queue* data structure, in which the entities in the collection are kept in order and the principal operations on the collection are the addition of entities to the rear terminal position and removal of entities from the front terminal position.

This makes the queue a First-In-First-Out (FIFO) data structure, where the first element added to the queue will be the first one to be removed.

We use this data structure to store what values of the x vector are the next to be tested, to check if the solution is feasible.

8.1.2 BalasAlgorithm.cpp

This is the main class of the program that includes all the functionalities that we are going to use to set the algorithm.

Among the most important functions we have the following:

- **Reading the Input File.**

  The program has the function *void readFromFile(char *nfile)* that reads the information of the input data file, and store it in data structures that correspond to the c vector, A matrix, b vector and equalities matrix.

- **Conversion.**
  The function is *void conversion()* and it has as aim converting the data inputs into the standard form.
- **Setting the values of the x vector.**

  For setting the values of x vector in each iteration we use two functions:

  bool createVariablesOne(bool* input, int index)* that sets the value of the x_{index} to 1 and

  bool createVariablesZero(bool* input, int index)* that sets the value of the x_{index} to 0 and the value of $x_{index+1}$ to 1.

- **Constraint Satisfied.**

  We have the function *int constraintSatisfied(int constraint)* to check if the constraint indicated by param is satisfied for the values of x vector in each iteration.

- **Calculating Objective Function.**

  For this purpose we have the method *double calculateZ()* that in each iteration calculates de value of Z in case it is a feasible solution and store it in a data structure to keep this value.

- **Pruning.**

  The function *bool prune(int index)* returns if the algorithm should continue through the current branch, from the index indicated by param.

- **Balas Algorithm.**

  The function *int balasAlgorithm()* represents the functionality of the Balas Algorithm, using the functions explained before.

9 Improving the processing speed of the program

We have to consider that the values that the x vector can take are binary values, 0 or 1. We can capitalize on it when we want to save processing time.

9.1 Operations Required

The first important thing is that the only operations that we need are additions and subtractions. When we calculate the value of the objective function or we check if the values of the x vector satisfy a constraint, we only need these two operations.

It is an advantage taking in account the computational complexity of each operation compared with the multiplication. In the next tableau we can see the computational complexity of each operation:

Operation	Input	Complexity
Addition	Two n-digit numbers	$\Theta(n)$
Subtraction	Two n-digit numbers	$\Theta(n)$
Multiplication	Two n-digit numbers	$O(n^2)$

9.1.1 Example

Suppose we have x vector as $(0,1,0,1)$ and we have to check if it satisfies the constraint

$$2x_1 + 3x_2 - 5x_3 + x_4 <= 7$$

As we can see x_1 and x_4 are 0. Hence, the only operations we have to do are with x_2 and x_4 doing an addition of $3 + 1$

Thus, we see that in contrast to integer linear programming where we have to use multiplications, in binary integer programming we only have to use addition and multiplication.

9.2 Pointers

One of the challenges that we have face in the project is trying to test the program with m and n taking high values. It entails a high computational charge, making a difference in hours for the program to find the best solution.

For that reason we have use pointers to cover the A matrix when we have to check if the x vector satisfies the constraints of the problem.

A lot of problems such as the *96 variables Diet Problem* have many values in the A matrix that are 0. Specifically, in the *96 variables Diet Problem,* the A matrix of dimension 39 x 96 has 1941 values equal to 0.

We can take advantage of this because we can save a lot of time covering the data structure that contains the A matrix. In order to do this, we use a 2 dimensions array of pointers of m rows that indicates what values in the A matrix are different from 0. With this array of pointers we save a lot of computational time.

9.2.1 Example

If we have the A matrix 4x4:

$$\begin{pmatrix} 4 & 0 & 5 & 3 \\ 3.5 & 7 & 8 & 0 \\ 2 & 0 & 1.5 & 0 \\ 0 & 4 & 0 & 9.3 \end{pmatrix}$$

We see that the first row has the positions a_1, a_3 and a_4 different from 0. In the second row they are the positions a_1, a_2 and a_3. In the third row they are a_1 and a_3. The last row has a_2 and a_4 different from 0.

Then the array of pointers contains the index of the positions different from 0:

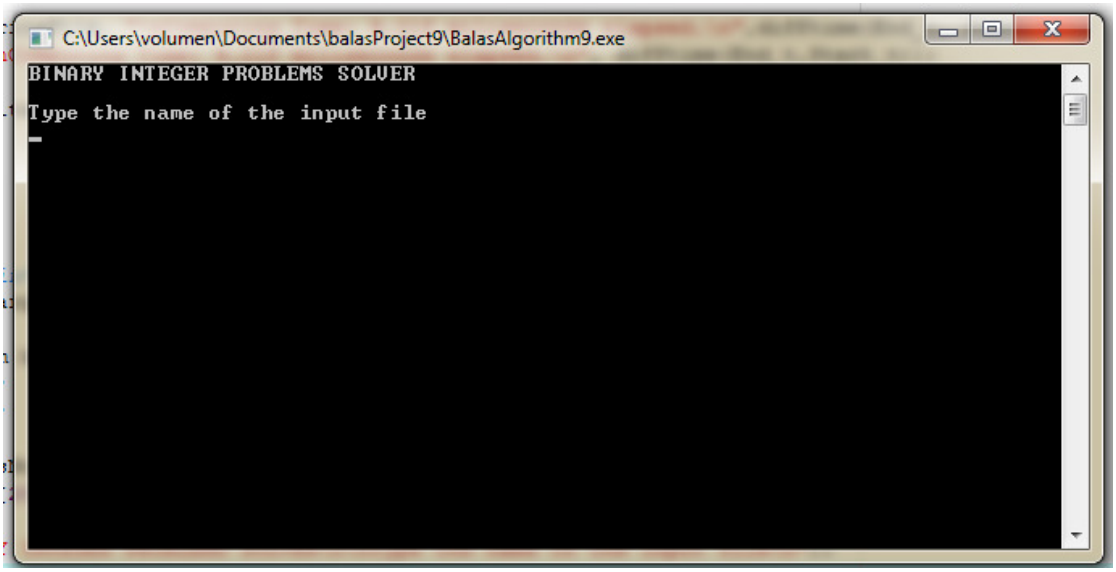
1	3	3
1	2	3
1	3	
2	4	

Thus, we have to cover 10 positions of the array of pointers instead of to cover 4x4=16 positions of the A matrix.

10 User's Guide

The program consists of an executable written in C++ which reads an input file with the data of the integer binary problem. When clicking the executable a console window appears with these characteristics:

- The first line will be the name of the program "Binary Integer Problems Solver".
- The next line it asks one to indicate the name of the input file.



10.1 Format of the input file

The format of the input file follows this structure:

```
m n

A matrix

b vector

min or max

c vector

Equalities vector, that should be as long as the b vector, and should
have for each constraint a 0 if it is <= or 1 if it is >=

F that should be 1 to show feasible solutions or 0 otherwise

P that should be 1 to show the internal operation of the algorithm or
0 otherwise.
```

For example, if we have the following problem:

Max 10x1 -7x2 + x3 -12x4 + 2x5 + 8x6 -3x7 –x8 +5x9 + 3x10

Subject to

3x1+ 12x2 -8x3 -1x4 -7x9 +2x10 >= -8

x2+10x3+ 5x5 –x6+ 7x7+ x8<= 13

-5x1 -3x2+ x3 -2x8 -x10 <= -6

5x1+ 3x2 –x3 + 2x8 +x9 >= -6

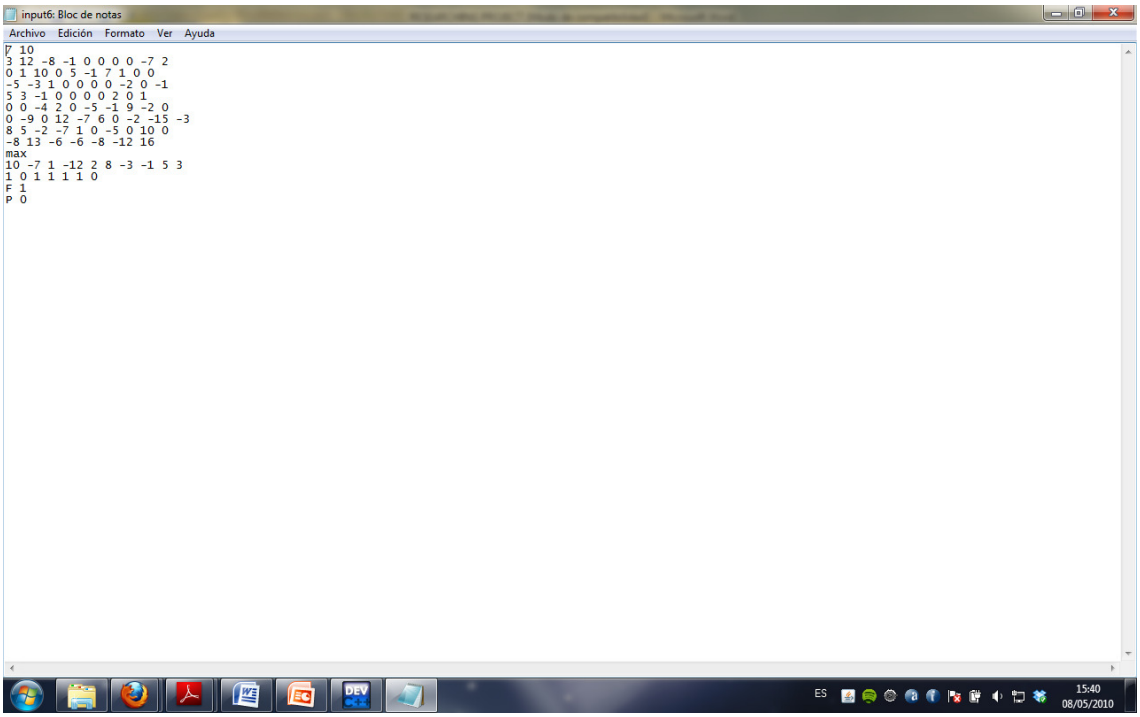
-4x3 +2x4 -5x6 –x7+ 9x8 -2x9 >=-8

-9x2+ 12x4 -7x5+ 6x6 -2x8 -15x9 -3x10 <=-12

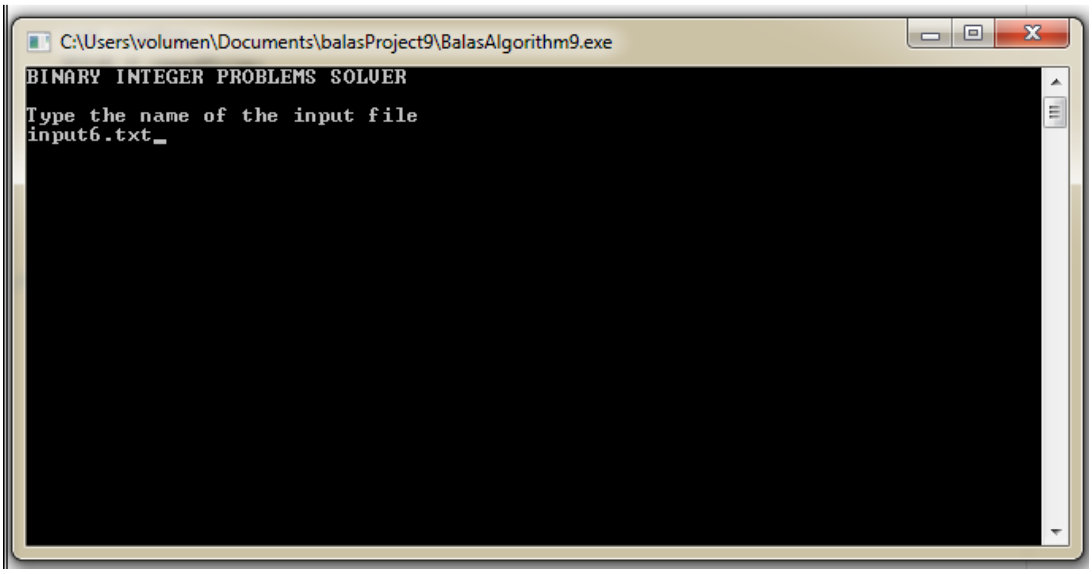
8x1+ 5x2 -2x3 -7x4+x5 -5x7+ 10x9<= 16

The input file would be as follows

```
7 10
3 12 -8 -1 0 0 0 0 -7 2
0 1 10 0 5 -1 7 1 0 0
-5 -3 1 0 0 0 0 -2 0 -1
5 3 -1 0 0 0 0 2 0 1
0 0 -4 2 0 -5 -1 9 -2 0
0 -9 0 12 -7 6 0 -2 -15 -3
8 5 -2 -7 1 0 -5 0 10 0
-8 13 -6 -6 -8 -12 16
max
10 -7 1 -12 2 8 -3 -1 5 3
1 0 1 1 1 1 0
F (0 or 1)
P (0 or 1)
```



The next step is to indicate the name of the input file in the program.

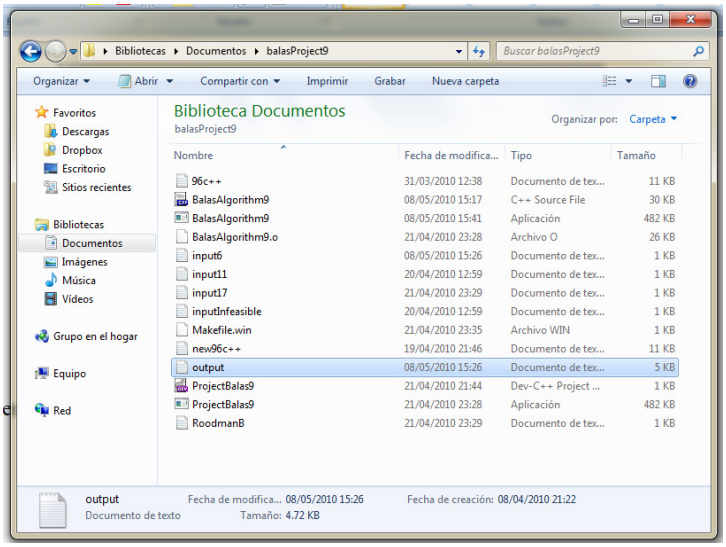


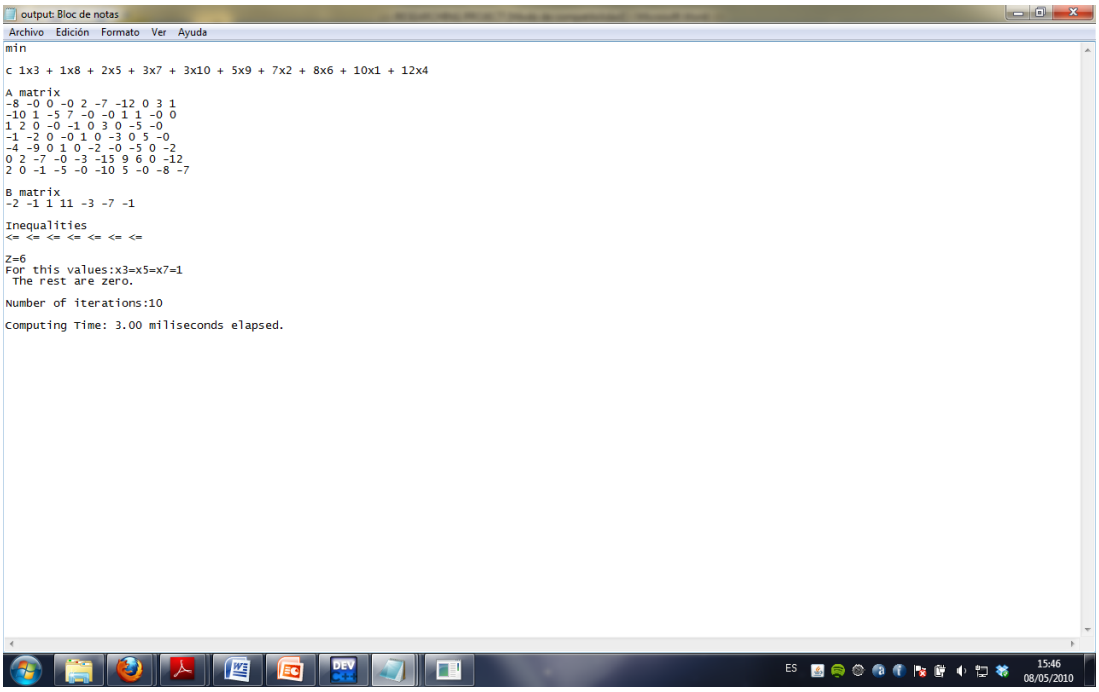
When we execute the program an output file is generated with the name *output.txt*

10.2 Format of the Output File

The output file has the following format:

```
Data of the problem

[Internal Operation of the program]

[Feasible Solutions]

Optimal Solution

Number of iterations

Computing Time
```


Bibliotecas ▸ Documentos ▸ balasProject9
Buscar balasProject9
Organizar
Abrir
Compartir con
Imprimir
Grabar
Nueva carpeta
Favoritos
Descargas
Dropbox
Escritorio
Sitios recientes
Bibliotecas
Documentos
Imágenes
Música
Vídeos
Grupo en el hogar
Equipo
Red
Biblioteca Documentos
balasProject9
Organizar por: Carpeta
Nombre
Fecha de modifica...
Tipo
Tamaño
96c++ 31/03/2010 12:38 Documento de tex... 11 KB
BalasAlgorithm9 08/05/2010 15:17 C++ Source File 30 KB
BalasAlgorithm9 08/05/2010 15:41 Aplicación 482 KB
BalasAlgorithm9.o 21/04/2010 23:28 Archivo O 26 KB
input6 08/05/2010 15:26 Documento de tex... 1 KB
input11 20/04/2010 12:59 Documento de tex... 1 KB
input17 21/04/2010 23:29 Documento de tex... 1 KB
inputInfeasible 20/04/2010 12:59 Documento de tex... 1 KB
Makefile.win 21/04/2010 23:35 Archivo WIN 1 KB
new96c++ 19/04/2010 21:46 Documento de tex... 11 KB
output 08/05/2010 15:26 Documento de tex... 5 KB
ProjectBalas9 21/04/2010 21:44 Dev-C++ Project ... 1 KB
ProjectBalas9 21/04/2010 23:28 Aplicación 482 KB
RoodmanB 21/04/2010 23:29 Documento de tex... 1 KB
output
Documento de texto
Fecha de modifica... 08/05/2010 15:26
Tamaño: 4.72 KB
Fecha de creación: 08/04/2010 21:22


output: Bloc de notas
Archivo Edición Formato Ver Ayuda
min
c 1x3 + 1x8 + 2x5 + 3x7 + 3x10 + 5x9 + 7x2 + 8x6 + 10x1 + 12x4
A matrix
-8 -0 0 -0 2 -7 -12 0 3 1
-10 1 -5 7 -0 -0 1 1 -0 0
1 2 0 -0 -1 0 3 0 -5 -0
-1 -2 0 -0 1 0 -3 0 5 -0
-4 -9 0 1 0 -2 -0 -5 0 -2
0 2 -7 -0 -3 -15 9 6 0 -12
2 0 -1 -5 -0 -10 5 -0 -8 -7
B matrix
-2 -1 1 11 -3 -7 -1
Inequalities
<= <= <= <= <= <= <=
Z=6
For this values:x3=x5=x7=1
The rest are zero.
Number of iterations:10
Computing Time: 3.00 miliseconds elapsed.
ES
15:46
08/05/2010

Hence, for our problem, the output is

```
min
c 1x3 + 1x8 + 2x5 + 3x7 + 3x10 + 5x9 + 7x2 + 8x6 + 10x1 + 12x4
A matrix
-8 -0 0 -0 2 -7 -12 0 3 1
-10 1 -5 7 -0 -0 1 1 -0 0
1 2 0 -0 -1 0 3 0 -5 -0
-1 -2 0 -0 1 0 -3 0 5 -0
-4 -9 0 1 0 -2 -0 -5 0 -2
0 2 -7 -0 -3 -15 9 6 0 -12
2 0 -1 -5 -0 -10 5 -0 -8 -7
B matrix
-2 -1 1 11 -3 -7 -1
Inequalities
<= <= <= <= <= <= <=
min Z=6
For:x3=x5=x7=1
The remaining variables are zero.
Number of iterations:10
Computing Time: 3.00 milliseconds elapsed.
```

11 Datasets tested

In this section we are going to show the different datasets that have been used to test the program. We are going to show five different datasets that go from the easy problems with seven variables and ten constraints to problems more complicated like the Diet Problem, with 96 variables and 39 constraints.

11.1 Problem 1 (Balas)

This is a problem with 10 variables and 7 constraints taken from the article of Egon Balas (1965). The optimum is for Z=6 for $x_3=x_5=x_7=1$.

11.1.1 Input file

The name of the file is *problemBalas.txt*

```
7 10
3 12 -8 -1 0 0 0 0 -7 2
0 1 10 0 5 -1 7 1 0 0
-5 -3 1 0 0 0 0 -2 0 -1
5 3 -1 0 0 0 0 2 0 1
0 0 -4 2 0 -5 -1 9 -2 0
0 -9 0 12 -7 6 0 -2 -15 -3
8 5 -2 -7 1 0 -5 0 10 0
-8 13 -6 -6 -8 -12 16
max
10 -7 1 -12 2 8 -3 -1 5 3
1 0 1 1 1 1 0
F 0
P 0
```

11.1.2 Output file

```
min

c 1x3 + 1x8 + 2x5 + 3x7 + 3x10 + 5x9 + 7x2 + 8x6 + 10x1 + 12x4

A matrix

-8 -0 0 -0 2 -7 -12 0 3 1

-10 1 -5 7 -0 -0 1 1 -0 0

1 2 0 -0 -1 0 3 0 -5 -0

-1 -2 0 -0 1 0 -3 0 5 -0

-4 -9 0 1 0 -2 -0 -5 0 -2

0 2 -7 -0 -3 -15 9 6 0 -12

2 0 -1 -5 -0 -10 5 -0 -8 -7

B matrix

-2 -1 1 11 -3 -7 -1

Inequalities

<= <= <= <= <= <= <=

min Z=6

For:x3=x5=x7=1

The remaining variables are zero.

Number of iterations:10

Computing Time: 2.00 milliseconds elapsed.
```

11.2 Problem 2 (Roodman)

Roodman's problem is taken from the article ROODMAN, G. M., "Postoptimality Analysis in Zero-One Programming by Implicit Enumeration" (1972). The problem consists of 8 variables and 3 constraints.

The optimum is Z=6 for x_3=x_5=1.

11.2.1 Input File

The name of the file is *problemRoodman.txt*

```
3 8

-2 0 0 2 -6 1 -1 2

-4 11 -11 -7 4 3 -5 1

0 1 1 1 -1 -2 0 1

-5 -6 0

min

2 5 5 6 4 1 8 1

0 0 0

F 0

P 0
```

11.2.2 Output File

min

c 1x6 + 1x8 + 2x1 + 4x5 + 5x2 + 5x3 + 6x4 + 8x7

A matrix

1 2 -2 -6 0 0 2 -1

3 1 -4 4 11 -11 -7 -5

-2 1 0 -1 1 1 1 0

B matrix

-5 -6 0

Inequalities

<= <= <=

min Z=9

For:x5=x3=1

The remaining variables are zero.

Number of iterations: 8

Computing Time: 0.00 milliseconds elapsed.

11.3 Problem 3 (Diet problem)

One of the problems that we have used to test the program is the Diet Problem. The goal of this problem is to find the cheapest combination of foods that will satisfy all the daily nutritional requirements of a person. For the complicated model see Sufahani (2010).

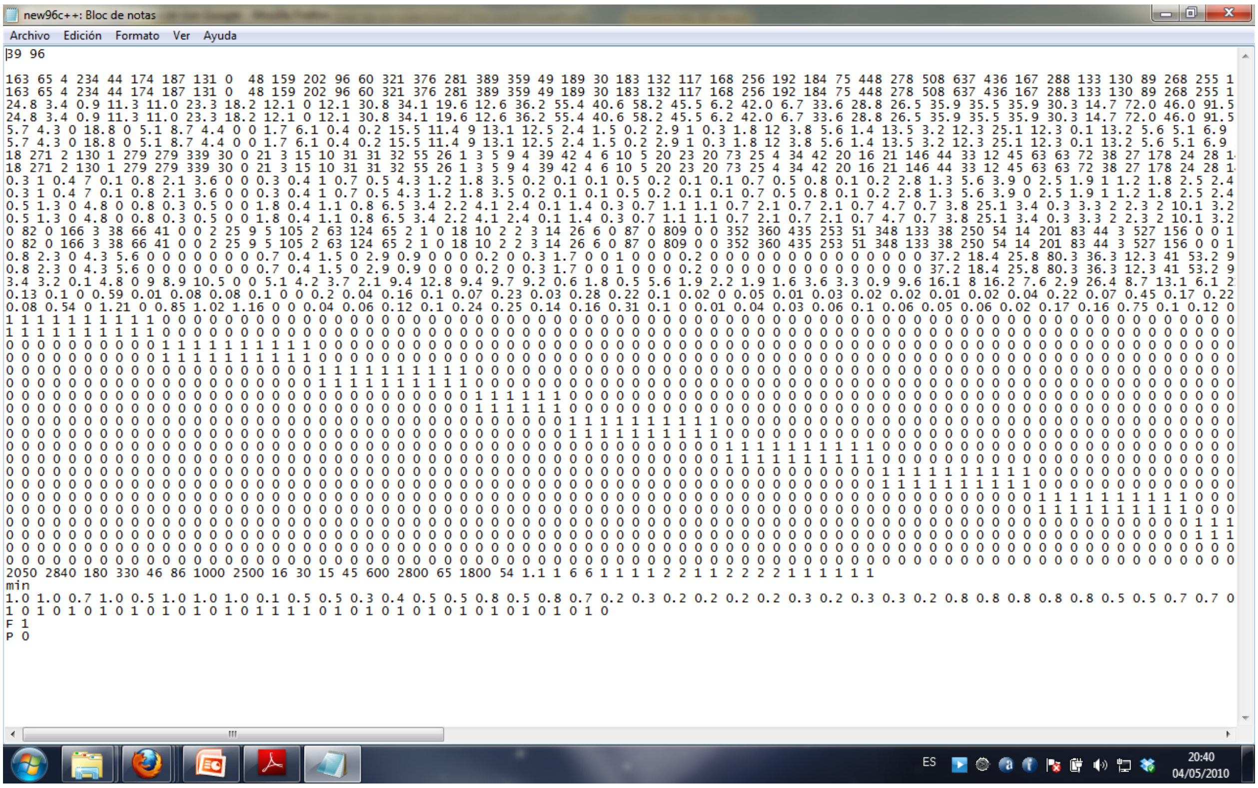
The problem is formulated as a linear program with 96 variables and 39 constraints, where the objective is to minimize cost and meet constraints which require that human nutritional needs are satisfied.

The optimum for this problem is min Z=8 for

x9=x20=x22=x47=x51=x58=x60=x68=x90=x5=x10=x36=x37=x3=x34=x79=x2=x8=1

11.3.1 Input file

The name of the file is *problemDiet.txt*



```
39 96

163 65 4 234 44 174 187 131 0  48 159 202 96 60 321 376 281 389 359 49 189 30 183 132 117 168 256 192 184 75 448 278 508 637 436 167 288 133 130 89 268 255 1
163 65 4 234 44 174 187 131 0  48 159 202 96 60 321 376 281 389 359 49 189 30 183 132 117 168 256 192 184 75 448 278 508 637 436 167 288 133 130 89 268 255 1
24.8 3.4 0.9 11.3 11.0 23.3 18.2 12.1 0 12.1 30.8 34.1 19.6 12.6 36.2 55.4 40.6 58.2 45.5 6.2 42.0 6.7 33.6 28.8 26.5 35.9 35.5 35.9 30.3 14.7 72.0 46.0 91.5
24.8 3.4 0.9 11.3 11.0 23.3 18.2 12.1 0 12.1 30.8 34.1 19.6 12.6 36.2 55.4 40.6 58.2 45.5 6.2 42.0 6.7 33.6 28.8 26.5 35.9 35.5 35.9 30.3 14.7 72.0 46.0 91.5
5.7 4.3 0 18.8 0 5.1 8.7 4.4 0 0 1.7 6.1 0.4 0.2 15.5 11.4 9 13.1 12.5 2.4 1.5 0.2 2.9 1 0.3 1.8 12 3.8 5.6 1.4 13.5 3.2 12.3 25.1 12.3 0.1 13.2 5.6 5.1 6.9
5.7 4.3 0 18.8 0 5.1 8.7 4.4 0 0 1.7 6.1 0.4 0.2 15.5 11.4 9 13.1 12.5 2.4 1.5 0.2 2.9 1 0.3 1.8 12 3.8 5.6 1.4 13.5 3.2 12.3 25.1 12.3 0.1 13.2 5.6 5.1 6.9
18 271 2 130 1 279 279 339 30 0 21 3 15 10 31 31 32 55 26 1 3 5 9 4 39 42 4 6 10 5 20 23 20 73 25 4 34 42 20 16 21 146 44 33 12 45 63 63 72 38 27 178 24 28 1
18 271 2 130 1 279 279 339 30 0 21 3 15 10 31 31 32 55 26 1 3 5 9 4 39 42 4 6 10 5 20 23 20 73 25 4 34 42 20 16 21 146 44 33 12 45 63 63 72 38 27 178 24 28 1
0.3 1 0.4 7 0.1 0.8 2.1 3.6 0 0 0.3 0.4 1 0.7 0.5 4.3 1.2 1.8 3.5 0.2 0.1 0.1 0.5 0.2 0.1 0.1 0.7 0.5 0.8 0.1 0.2 2.8 1.3 5.6 3.9 0 2.5 1.9 1 1.2 1.8 2.5 2.4
0.3 1 0.4 7 0.1 0.8 2.1 3.6 0 0 0.3 0.4 1 0.7 0.5 4.3 1.2 1.8 3.5 0.2 0.1 0.1 0.5 0.2 0.1 0.1 0.7 0.5 0.8 0.1 0.2 2.8 1.3 5.6 3.9 0 2.5 1.9 1 1.2 1.8 2.5 2.4
0.5 1.3 0 4.8 0 0.8 0.3 0.5 0 0 1.8 0.4 1.1 0.8 6.5 3.4 2.2 4.1 2.4 0.1 1.4 0.3 0.7 1.1 1.1 0.7 2.1 0.7 2.1 0.7 4.7 0.7 3.8 25.1 3.4 0.3 3.3 2 2.3 2 10.1 3.2
0.5 1.3 0 4.8 0 0.8 0.3 0.5 0 0 1.8 0.4 1.1 0.8 6.5 3.4 2.2 4.1 2.4 0.1 1.4 0.3 0.7 1.1 1.1 0.7 2.1 0.7 2.1 0.7 4.7 0.7 3.8 25.1 3.4 0.3 3.3 2 2.3 2 10.1 3.2
0 82 0 166 3 38 66 41 0 0 2 25 9 5 105 2 63 124 65 2 1 0 18 10 2 2 3 14 26 6 0 87 0 809 0 0 352 360 435 253 51 348 133 38 250 54 14 201 83 44 3 527 156 0 0 1
0 82 0 166 3 38 66 41 0 0 2 25 9 5 105 2 63 124 65 2 1 0 18 10 2 2 3 14 26 6 0 87 0 809 0 0 352 360 435 253 51 348 133 38 250 54 14 201 83 44 3 527 156 0 0 1
0.8 2.3 0 4.3 5.6 0 0 0 0 0 0 0.7 0.4 1.5 0 2.9 0.9 0 0 0 0.2 0 0.3 1.7 0 0 1 0 0 0 0.2 0 0 0 0 0 0 0 0 0 0 0 0 0 37.2 18.4 25.8 80.3 36.3 12.3 41 53.2 9
0.8 2.3 0 4.3 5.6 0 0 0 0 0 0 0.7 0.4 1.5 0 2.9 0.9 0 0 0 0.2 0 0.3 1.7 0 0 1 0 0 0 0.2 0 0 0 0 0 0 0 0 0 0 0 0 0 37.2 18.4 25.8 80.3 36.3 12.3 41 53.2 9
3.4 3.2 0.1 4.8 0 9 8.9 10.5 0 0 5.1 4.2 3.7 2.1 9.4 12.8 9.4 9.7 9.2 0.6 1.8 0.5 5.6 1.9 2.2 1.9 1.6 3.6 3.3 0.9 9.6 16.1 8 16.2 7.6 2.9 26.4 8.7 13.1 6.1 2
0.13 0.1 0 0.59 0.01 0.08 0.08 0.1 0 0 0.2 0.04 0.16 0.1 0.07 0.23 0.03 0.28 0.22 0.1 0.02 0 0.05 0.01 0.03 0.02 0.02 0.01 0.02 0.04 0.22 0.07 0.45 0.17 0.22
0.08 0.54 0 1.21 0 0.85 1.02 1.16 0 0 0.04 0.06 0.12 0.1 0.24 0.25 0.14 0.16 0.31 0.1 0 0.01 0.04 0.03 0.06 0.1 0.06 0.05 0.06 0.02 0.17 0.16 0.75 0.1 0.12 0
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2050 2840 180 330 46 86 1000 2500 16 30 15 45 600 2800 65 1800 54 1.1 1 6 6 1 1 1 1 2 2 1 1 2 2 2 2 1 1 1 1 1 1
min
1.0 1.0 0.7 1.0 0.5 1.0 1.0 1.0 0.1 0.5 0.5 0.3 0.4 0.5 0.5 0.8 0.5 0.8 0.7 0.2 0.3 0.2 0.2 0.2 0.2 0.3 0.2 0.3 0.3 0.2 0.8 0.8 0.8 0.8 0.8 0.5 0.5 0.7 0.7 0
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
F 1
P 0
```

11.3.2 Output file

```
min Z=8

For:

x9=x20=x22=x47=x51=x58=x60=x68=x90=x5=x10=x36=x37=x3=x34=x79=x2=x8=1

The remaining variables are zero.

Computing Time: 119.00 milliseconds elapsed.
```

The optimal solution for this problem is Z=8, which is the same solution that we can get with other software as LPSolve or AMPL.

The computing time for the problem is 119 milliseconds. For AMPL it takes 0.039 seconds and for LPSolve 0.031 seconds, not so much difference taking into account that these type of software use presolving.

12 Conclusions

In a finite number of iterations, the additive algorithm yields either an optimal feasible solution, or the conclusion that the problem has no feasible solution at all.

The only operations required under the algorithm described above are additions and subtractions saving computational time.

The algorithm does not impose a heavy burden on the storage system of the computer.

The number of iterations depends on the characteristics of the problem.

13 References

Balas E. “An additive algorithm for solving linear programs with zero-one variables”. Operations Res. 13:517-46, 1965.

Sufahani Diet Problem (2010).

Richard Bronson Schaum's Outline of Operations Research (Schaum's Outline Series)

Alexander Schrijver Theory of Linear and Integer Programming

Hamdy A. Taha Integer Programming: Theory, Applications and Computations (Operations research in industrial engineering).

George Hadley Linear Programming.