GRADO UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE GRADO

ANÁLISIS DE SENSIBILIDAD DE SIMPLIFICACIONES EN PROBLEMAS DE OPTIMIZACIÓN LINEAL

Alumno: José María García-Mina Peñaranda

Director: Sara Lumbreras Sancho

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Análisis de sensibilidad de simplificaciones en problemas de optimización lineal.

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2023/24 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha

sido tomada de otros documentos está debidamente referenciada

Fdo.: José María García-Mina Peñaranda Fecha: 25/ 07/2024

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Sara Lumbreras Sancho Fecha: 25/07/2024

Executive Summary

Introduction

In the world of Operational Research (OR), linear optimization is one of the most used tools to solve problems. Linear Programming (LP) has been studied for more than a century, becoming the most extensively studied optimization problem.

Thanks to the versatility of this science, it applies to a wide range of different fields, from engineering to financial, going through social and environmental issues. When it comes to a real-life problem, the dimensions start getting difficult to manage, and even though there are algorithms which can solve them, the time that takes to reach an optimal solution is absurdly long. Here is where the simplification appears. In this paper, a sensibility analysis for a simplification operation is made, in order to see how the problems react to different levels of simplification, with the objective of being able to make a decision whether it is or it is not worth it. This will be possible with the help of three indexes and graphs of their evolution during the simplification process.

Pre-solve methods

Before starting to solve any LP, there exist some operations that simplify the problem. These operations are called pre-solved methods. The main objective of these methods is to reduce as much as possible the dimensions of the problem. In this paper, the 9 most popular methods are explained and illustrated with a simple example. The battery of problems used in the project has been pre-solved with these methods, before solving and applying the simplification operation.

Problems battery under study

The problems studied in this project is selected from the model library of the software GAMS, from where thirty LPs are randomly selected and afterwards pre-solved, solved and simplified. The battery is formed by problems from different fields such as: agricultural economics, management science and OR, stochastic programming, macro and micro economics and mathematics.

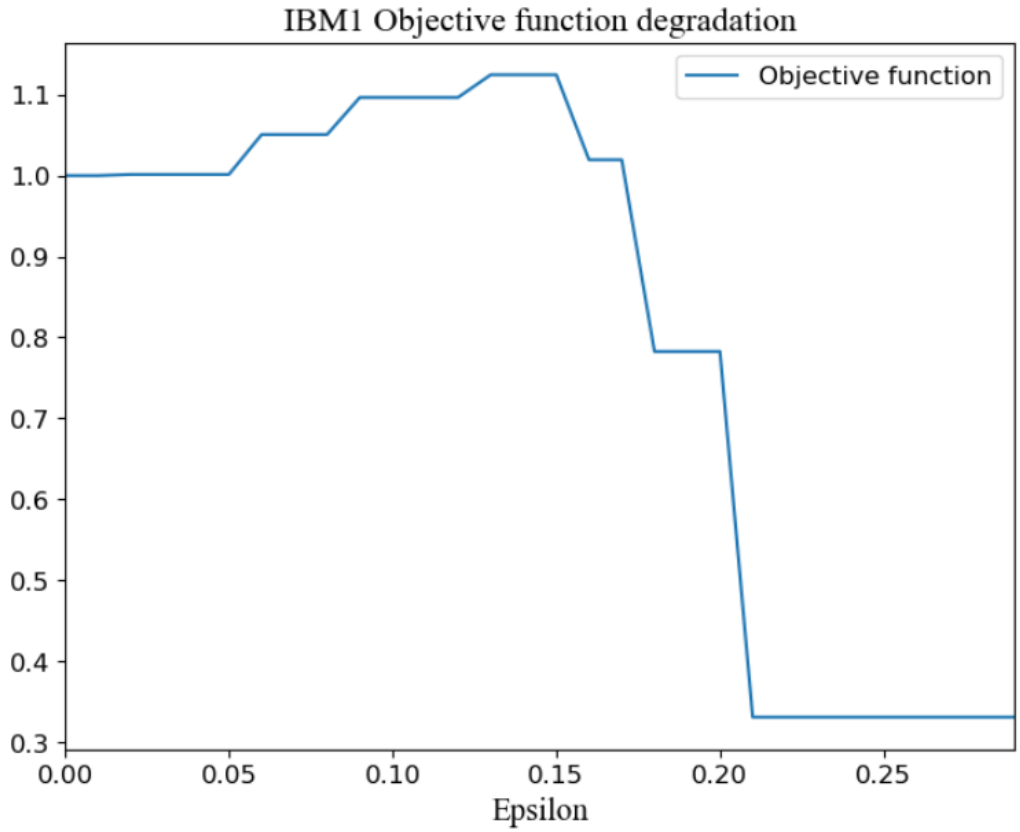
Sensibility analysis

Once the problems have been pre-solved, solved and simplified with the *sparsification*[1] operation, main data is uploaded into Jupyter notebook, the environment where the coding is made. The main data is:

- Epsilon: signification level that is used to modify the matrix A, if the element is less than epsilon (matrix A normalized) it takes 0 value.
- Objective_function: With the new matrix A (modified using epsilon), a new model is built. This is the value for the new objective function for each epsilon.
- Decision_variables: It contains the value of each decision variable for each epsilon in the new model built with the modified matrix A.
- Changed_indices: Elements that have been modified from the matrix A, for each value of epsilon.
- Constraint_violation: Taking the solution of the new model (modified matrix A) and putting it in the original model, the violation of each constraint is evaluated, for each epsilon value.
- of_original_decision: Taking the solution of the new model (modified matrix A) and putting it in the original model, the objective function is evaluated, for each epsilon value.
- time_required: It shows the time required to solved the problem for each epsilon value.

Using this data three indexes are calculated to measure the evolution of the objective function, the complexity and the infeasibility of the problem. These indexes are normalized, so comparisons between different problems are possible. For every value of epsilon, each index will have a value, so afterwards the graph of the evolution during the simplification will be made for every index. Here is an example (problem form the battery called IBM1) of the graph of the evolution of the objective function through the simplification process:

[1] Simplification operation used and applied by the Co-director of the project, Phillipe Vilaça Gomes.



Graphs like the one above will be obtained for the other two indexes too. With a function coded, the entire battery will be iterated with a loop. With this iteration, the three graphs will be obtained for every problem from the battery. Once that all the graphs are obtained, four main behaviours are visualized. The first behaviour consists of a notably improvement on the objective function value and in the complexity of the problem, but with the cost of the problem becoming extremely infeasible. The second behaviour is quite similar, but in this case the objective function worsens. For the complexity and the infeasibility happens the same as in the first one. The third type consists of the problems that do not see themselves affected until big epsilons. When this happens the objective function and complexity improve, but at the same time the problem becomes infeasible. For the three cases mention before, the sensibility analysis shows that it has no value the simplification in these kinds of problems. But, the four behaviour is the one where the objective function and the infeasibility index barely vary, while the complexity of the problem decreases. For this behaviour, the simplification operation is worth

it, because the problem will be simplified in a notably percentage, while keeping almost the same objective function value, and without becoming infeasible.

Conclusion

The main conclusions obtained with this project can be synthesized in two main points:

Firstly, each problem must be study and simplified individually, since it has been clearly obtained that there are different behaviours through the simplification operation. Even though most of the problems cannot be simplified, because they become infeasible, there are some of them that are acceptable for the simplification, reaching an easier solution without affecting the infeasibility or the objective function value. This can be translated into the use of less variables and/or constraints.

The second point is that with the sensibility analysis defined in the project, any project could be easily analysed and classified in one of the four main behaviours, just by looking at three graphs. With this information, decisions towards simplifying or not the LP can be made.

Author: García-Mina Peñaranda, José María.

Director: Lumbreras Sancho, Sara.

Co-director: Villaça Gomes, Phillipe.

Abstract

Linear optimization is an essential tool when facing complex problems. Thanks to its ability to adapt to all types of challenges across various fields of application, it is highly prevalent in the daily activities of any profession. Because real-life problems complexity, due to the number of variables and constraints, simplification is key. Adding to the traditional pre-solved methods, simplification operations are in constant innovation, in order to make these problems easier. With this project, a sensibility analysis is made, to see the reaction of a battery of problems to a simplification operation called *sparsification,* which will be done for different signification levels (epsilons) from less to more simplified. The study will use a battery of LPs from the optimization software GAMS.

First of all, three main indexes are calculated, all of them normalized so it is possible making comparisons between problems. The first one measures the objective function value, the second one measures the infeasibility and the last one the complexity of the problem. The three will be calculated for every epsilon, and afterwards, the graphs for the three of them will be obtained.

Once the graphs are obtained, the analysis starts. It consists of searching for similar reactions and patterns in the different problems. Four main behaviours are identified in the analysis, and only one of them shows that the simplification operation is worth it.

Key words: simplification, sensibility, linear programming, optimization, behaviours, infeasibility, complexity

Autor: García-Mina Peñaranda, José María.

Director. Lumbreras Sancho, Sara.

Co-director: Villaça Gomes, Phillipe.

Resumen

La optimización lineal es una herramienta esencial al enfrentar problemas complejos. Gracias a su capacidad para adaptarse a todo tipo de desafíos en diversos campos de aplicación, es altamente prevalente en las actividades diarias de cualquier profesión. Debido a la complejidad de los problemas de la vida real, por el número de variables y restricciones, la simplificación es clave. Además de los métodos tradicionales pre-resueltos, las operaciones de simplificación están en constante innovación para hacer estos problemas más manejables. Con este proyecto, se realiza un análisis de sensibilidad para observar la reacción de una batería de problemas a una operación de simplificación llamada esparsificación, que se llevará a cabo para diferentes niveles de significación (épsilons), desde menos hasta más simplificado. El estudio utilizará una batería de PLs del software de optimización GAMS.

En primer lugar, se calculan tres índices principales, todos ellos normalizados para que sea posible hacer comparaciones entre problemas. El primero mide el valor de la función objetivo, el segundo mide la inviabilidad y el último la complejidad del problema. Los tres se calcularán para cada épsilon y, posteriormente, se obtendrán los gráficos de los tres.

Una vez obtenidos los gráficos, comienza el análisis. Consiste en buscar reacciones y patrones similares en los diferentes problemas. En el análisis se identifican cuatro comportamientos principales, y solo uno de ellos muestra que la operación de simplificación merece la pena.

Palabras clave: simplificación, sensibilidad, programación lineal, optimización, comportamientos, infactibilidad, complejidad.

Content table

Illustration index

1. Introduction

Linear optimization is an essential tool when facing complex problems. Thanks to its ability to adapt to all types of challenges across various fields of application, it is highly prevalent in the daily activities of any profession. One of the characteristics of these problems is the large number of variables and constraints, which makes solving them an arduous task. Because of this, the simplification of linear programming is constantly innovating and trying to find a way to make this science easier. Numerous models of simplifying problems exist, and with this analysis it will be possible to evaluate the impact of the simplification to complex linear programming problems.

The project explains basic linear programming knowledge, shows the ten most important pre-solve methods that are used in simplification operations and then enters the sensibility analysis, ending with the conclusions obtained by the analysis. All the codes used for the operations and obtaining the graphs are in the annex.

Firstly, sixty problems will be solved and simplify by the *sparsification* operation, saving the results of this operations in a json file. This simplification solves the problem for 30 different levels of signification (epsilons), being the first one the original problem and the last one the most simplified. To read and study the data the open-source web application Jupyter notebook is used, programming in Python. The objective of this study is to visualize graphically the evolution of the LPs problems and how they react to the simplification.

The main indicators for the project are the normalized objective function, an unfeasibility index and a complexity index, that are calculated for each value of epsilon. First this three are calculated just for a small sample of problems, and then a function is program with all the operations needed, to then iterate every problem and be able to observe the graphs and reach conclusions with the entire battery.

2. Linear programming

The Linear Programming (LP) problem is arguably the most significant and extensively studied optimization problem. A multitude of real-world issues can be formulated as Linear Programming problems (LPs). LP involves the process of minimizing or maximizing a linear objective function subject to a ser of linear equality and/or inequality constraints.

The structure of a LP problem in standard form is the following:

- Objective function: min $z = c^T x$

- Constraints: s.t $Ax \geq b$

$x \geq 0$

Where x is the vector of decision variables, A is the matrix of restrictions, c is the coefficients vector of the objective function and b is the right-hand side vector. Furthermore, $A \in \mathbb{R}^{m \times n}$, $(c, x) \in \mathbb{R}^n$ and T denotes transposition, and that the linear system $Ax = b$ is consistent. With the word min it is shown that the problem is to minimize de objective function. In order to be a linear programming problem, both the objective function and the constraints must be linear.

In addition, from every LP we can obtain the dual problem. Both problems keep a close relation between them. The canonical form of the dual problem is the following:

- Objective function: max $z = b^T w$

- Constraints: s.t $A^T w \leq c$

$w \geq 0$

The easiest problems can be solved graphically, while for the more complex ones the most popular way of approaching these problems is the simplex algorithm. The

algorithm starts with a first feasible solution and moves towards an adjacent solution until the optimal is reached.

3. Pre-solve methods.

3.1. Introduction

Pre-solve methods play a crucial role in solving linear programming (LP) problems by reducing their size and determining if they are unbounded or infeasible. These methods are applied before an LP algorithm to: remove redundant constraints, fix certain variables, adjust bounds on individual structural variables, and decrease the number of variables and constraints through eliminations.

Nine pre-solve methods used before executing an LP algorithm are going to be shown: eliminating zero rows, eliminating zero columns, eliminating singleton equality constraints, eliminating kton equality constraints, eliminating singleton inequality constraints, eliminating dual singleton inequality constraints, eliminating implied free singleton columns, eliminating redundant columns, eliminating implied bounds on rows, eliminating redundant rows, and ensuring the coefficient matrix is structurally full rank.

The following LP problem in canonical form is considered:

$$\begin{aligned} \min \quad & z = c^T x \\ \text{s.t} \quad & \underline{b} \leq Ax \leq \overline{b} \\ & \underline{x} \leq x \leq \overline{x} \end{aligned}$$

Where $c, x \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, \underline{x} = 0, \overline{x} = \infty, \underline{b} = (\mathbb{R} \cup \{-\infty\})^m, \overline{b} = (\mathbb{R} \cup \{+\infty\})^m$, and T denotates transposition. Let A_i be the ith row of matrix A and A_j be the jth column of the matrix A.

3.2. Methods[2]

3.2.1. Eliminate zero rows.

A row in the coefficient matrix A is considered an empty row if all the coefficients in that row are zero. A zero row can be expressed as:

$$\underline{b_i} \leq A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n \leq \overline{b_i}$$

A constraint of this type maybe redundant or may state that the LP problem is infeasible. All possible cases are distinguished in the following theorem:

For each empty row we distinguished the following cases:

1. $A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n \leq \overline{b_i}\ and\ \overline{b_i} \geq 0$: The constraint is redundant and can be deleted.
2. $A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n \leq \overline{b_i}\ and\ \overline{b_i} < 0$: The LP problem is infeasible.
3. $A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n \geq \underline{b_i}\ and\ \underline{b_i} \leq 0$: The constraint is redundant and can be deleted.
4. $A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n \geq \underline{b_i} and\ \underline{b_i} > 0$: The LP problem is infeasible.
5. $A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n \geq \underline{b_i} = \overline{b_i} = b_i\ and\ b_i = 0$: The constraint is redundant and can be deleted.

[2] Methods and examples obtained from Ploskas, N., & Samaras, N. (n.d.). Springer Optimization and its applications.

6. $A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n \geq \underline{b_i} = \overline{b_i} = b_i \ and \ b_i \neq 0$: The LP problem is infeasible.

With the next illustrative example, we can see the demonstration of the pre-solve method that eliminates zero rows.

The LP problem that will be presolved is the following:

$$\begin{aligned}
\min \quad & z = -x_1 + 4x_2 + 5x_3 - 2x_4 - 8x_5 + 2x_6 \\
\text{s.t} \quad & 2x_1 - 3x_2 + 3x_5 + x_6 \leq 9 \ (1) \\
& -x_1 + 3x_2 + 2x_3 - x_5 - 2x_6 \geq 1 \ (2) \\
& 0x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 \geq -5 \ (3) \\
& 7x_1 + 5x_2 + 2x_3 - 2x_5 + 4x_6 = 7 \ (4) \\
& 0x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 \geq -10 \ (5) \\
x_j \geq 0, \quad & (j = 1, 2, 3, 4, 5, 6)
\end{aligned}$$

The matrix notation is:

$$A = \begin{bmatrix} 2 & -3 & 0 & 0 & 3 & 1 \\ -1 & 3 & 2 & 0 & -1 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 5 & 2 & 0 & -2 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, c = \begin{bmatrix} -1 \\ 4 \\ 5 \\ -2 \\ -8 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 9 \\ 1 \\ -5 \\ 7 \\ -10 \end{bmatrix}, Eqin = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

We observe that all the elements of the third and fifth row are equal to zero. According to the third case of the previous section, the constraint is redundant and

can be deleted. Therefore, we can delete the third and fifth row of matrix A and the third and fifth element from vectors b and Eqin:

$$A = \begin{bmatrix} 2 & -3 & 0 & 0 & 3 & 1 \\ -1 & 3 & 2 & 0 & -1 & -2 \\ 7 & 5 & 2 & 0 & -2 & 4 \end{bmatrix}, c = \begin{bmatrix} -1 \\ 4 \\ 5 \\ -2 \\ -8 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 9 \\ 1 \\ 7 \end{bmatrix}, Eqin = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$

Finally, the equivalent LP problem after presolve is:

$$\begin{aligned} \min \quad & z = -x_1 + 4x_2 + 5x_3 - 2x_4 - 8x_5 + 2x_6 \\ \text{s.t} \quad & 2x_1 - 3x_2 + 3x_5 + x_6 \leq 9 \\ & -x_1 + 3x_2 + 2x_3 - x_5 - 2x_6 \geq 1 \\ & 7x_1 + 5x_2 + 2x_3 - 2x_5 + 4x_6 = 7 \\ x_j \geq 0, \quad & (j = 1, 2, 3, 4, 5, 6) \end{aligned}$$

3.2.2. Eliminate zero columns.

A column of the coefficient matrix A is considered an empty column if all the coefficients in that column are zero. A variable associated with such a column may either be redundant or indicate that the LP problem is unbounded. The following theorem distinguishes between these two scenarios:

1. $c_j \geq 0$: *The variable is redundant and can be deleted*

2. $c_j < 0$: *The LP problem is unbounded*

Through the next illustrative example, this method is demonstrated:

The LP problem that will be presolved is the following:

$$
\begin{aligned}
\min \quad & z = -x_1 + 4x_2 + 5x_3 + 2x_4 - 8x_5 + 2x_6 \\
\text{s.t} \quad & 2x_1 - 3x_2 + 3x_5 + x_6 \leq 9 \\
& -x_1 + 3x_2 - x_5 - 2x_6 \geq 1 \\
& x_1 \geq -5 \\
& 6x_1 + 5x_2 - 2x_5 + 4x_6 = 7 \\
& 3x_5 + 4x_6 \geq -10 \\
& x_j \geq 0, \quad (j = 1, 2, 3, 4, 5, 6)
\end{aligned}
$$

In matrix notation:

$$
A = \begin{bmatrix} 2 & -3 & 0 & 0 & 3 & 1 \\ -1 & 3 & 0 & 0 & -1 & -2 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 6 & 5 & 0 & 0 & -2 & 4 \\ 0 & 0 & 0 & 0 & 3 & 4 \end{bmatrix}, c = \begin{bmatrix} -1 \\ 4 \\ 5 \\ 2 \\ -8 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 9 \\ 1 \\ -5 \\ 7 \\ -10 \end{bmatrix}, Eqin = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}
$$

We observe that all the elements of the third and fourth column are equal to zero. According to the first case of the previous subsection, the variable is redundant and can be deleted. Therefore, we can delete both columns of matrix A, and the third and fourth elements from vector c. The presolved LP problem is now:

$$A = \begin{bmatrix} 2 & -3 & 3 & 1 \\ -1 & 3 & -1 & -2 \\ 1 & 0 & 0 & 0 \\ 6 & 5 & -2 & 4 \\ 0 & 0 & 3 & 4 \end{bmatrix}, c = \begin{bmatrix} -1 \\ 4 \\ 2 \\ -8 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 9 \\ 1 \\ -5 \\ 7 \\ -10 \end{bmatrix}, Eqin = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

And the equivalent LP problem after presolve is:

$$\begin{aligned} \min \quad & z = -x_1 + 4x_2 - 8x_5 + 2x_6 \\ \text{s.t} \quad & 2x_1 - 3x_2 + 3x_5 + x_6 \leq 9 \\ & -x_1 + 3x_2 - x_5 - 2x_6 \geq 1 \\ & x_1 \geq -5 \\ & 6x_1 + 5x_2 - 2x_5 + 4x_6 = 7 \\ & 3x_5 + 4x_6 \geq -10 \\ & x_j \geq 0, \quad (j = 1, 2, 5, 6) \end{aligned}$$

3.2.3. Eliminate singleton equality constraints.

An equality row in the coefficient matrix A is considered a singleton row if and only if it contains exactly one nonzero coefficient. A singleton equality row can be expressed as follows:

$$A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n = b_i$$

Where $A_{ik} \neq 0 \wedge A_{ij} = 0, i = 1, 2, \ldots, m,\ j = 1,2, \ldots, n,\ and\ j \neq k.$ The constraint can be rewritten as:

$$A_{ik}x_k = b_i$$

Therefore, the value of x_k is fixed at b_i/A_{ik}. A constraint of this type may either be redundant or indicate that the linear programming (LP) problem is infeasible. We can identify the following scenarios:

1. $x_k \geq 0$: Row i and column k are redundant and can be deleted.

2. $x_k < 0$: The LP problem is infeasible.

If $x_k \geq 0$ we replace x_k to all constraints:

$$\overline{b} = b - x_k A_{.k}$$

If $c_k \neq 0$, then a constant term of the objective function is computed as:

$$c_0 = c_0 - c_k * (\frac{b_i}{A_{ik}})$$

After making that replacement, row i is deleted from matrix A, element i is removed from vectors b and Eqin, column k is deleted from matrix A, and element k is removed from vector c. It is common for a new singleton equality row to appear after eliminating the previous one. Therefore, the current presolve method continues until no additional singleton equality rows are present.

With the next example we can see a demonstration:

$$\begin{aligned}
\text{min} \quad & z = -2x_1 + 4x_2 - 2x_3 + 2x_4 \\
\text{s.t} \quad & + 3x_3 = 6 \\
& 4x_1 - 3x_2 + 8x_3 - x_4 = 20 \\
& -3x_1 + 2x_2 - 4x_4 = -8 \\
& 4x_1 - x_3 = 18 \\
x_j \geq 0, \quad & (j = 1, 2, 3, 4)
\end{aligned}$$

In matrix notation:

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 4 & -3 & 8 & -1 \\ -3 & 2 & 0 & -4 \\ 4 & 0 & -1 & 0 \end{bmatrix}, c = \begin{bmatrix} -2 \\ 4 \\ -2 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 6 \\ 20 \\ -8 \\ 18 \end{bmatrix}, Eqin = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Initially, we begin by searching for equality constraints that contain only one nonzero element. We observe that, in the first equality constraint, all elements are zero except for the third element:

$$3x_3 = 6$$

So, the value for x_3:

$$3x_3 = \frac{6}{3} = 2$$

Following the first case of the previous subsection, the first row and the third column are redundant and can be deleted. We then update vector b:

$$\overline{b} = b - x_3 A_{.3} = \begin{bmatrix} 6 \\ 20 \\ -8 \\ 18 \end{bmatrix} - 2 \begin{bmatrix} 3 \\ 8 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ -8 \\ 20 \end{bmatrix}$$

$c_3 \neq 0$, so a constant term of the objective function is computed as:

$$c_0 = c_0 - (-2) * \left(\frac{6}{3}\right) = 0 + 4 = 4$$

Next, we delete the first row and the third column from matrix A, the first element from vectors b and Eqin, and the third element from vector c. The presolved LP problem is now:

$$A = \begin{bmatrix} 4 & -3 & -1 \\ -3 & 2 & -4 \\ 4 & 0 & 0 \end{bmatrix}, c = \begin{bmatrix} -2 \\ 4 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 4 \\ -8 \\ 20 \end{bmatrix}, Eqin = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, c_0 = 4$$

3.2.4. Eliminate singleton inequality constraints.

An inequality row in the coefficient matrix A is considered a singleton row if and only if it contains exactly one nonzero coefficient. A singleton inequality row can be expressed as follows:

$$\underline{b_i} \leq A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n \leq \overline{b_i}$$

Where $A_{ik} \neq 0 \wedge A_{ij} = 0, i = 1,2,\ldots,m,\ j = 1,2,\ldots,n,\ and\ j \neq k$. A constraint of this type may either be redundant or indicate that the linear programming (LP) problem is infeasible. All possible cases in the following theorem.

For each singleton inequality constraint, we distinguish the following cases:

1. Constraint type $\leq \overline{b_i}, A_{ik} > 0\ and\ \overline{b_i} < 0$: The LP is infeasible
2. Constraint type $\leq \overline{b_i}, A_{ik} < 0\ and\ \overline{b_i} > 0$: Row i is redundant and can be deleted.
3. Constraint type $\leq \overline{b_i}, A_{ik} > 0\ and\ \overline{b_i} = 0$: Row i and column k are redundant and can be deleted
4. Constraint type $\leq \overline{b_i}, A_{ik} < 0\ and\ \overline{b_i} = 0$: Row i is redundant and can be deleted.
5. Constraint type $\geq \underline{b_i}, A_{ik} > 0\ and\ \underline{b_i} < 0$: Row i is redundant and can be deleted
6. Constraint type $\geq \underline{b_i}, A_{ik} < 0\ and\ \underline{b_i} > 0$: The LP problem is infeasible.
7. Constraint type $\geq \underline{b_i}, A_{ik} > 0\ and\ \underline{b_i} = 0$: Row i is redundant and can be deleted
8. Constraint type $\geq \underline{b_i}, A_{ik} < 0\ and\ \underline{b_i} = 0$: Row i and column k are redundant and can be deleted

Throughout the next illustrative example this method is demonstrated.

$$
\begin{aligned}
\min \quad & z = -2x_1 + 4x_2 - 2x_3 + 2x_4 \\
\text{s.t} \quad & -3x_3 \leq 2 \\
& 4x_1 - 3x_2 + 8x_3 - x_4 = 20 \\
& -3x_1 + 2x_2 - 4x_4 \geq -8 \\
& -x_3 = 18 \\
& x_j \geq 0, \quad (j = 1, 2, 3, 4)
\end{aligned}
$$

In matrix notation:

$$
A = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 4 & -3 & 8 & -1 \\ -3 & 2 & 0 & -4 \\ 0 & 0 & -1 & 0 \end{bmatrix}, c = \begin{bmatrix} -2 \\ 4 \\ -2 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 2 \\ 20 \\ -8 \\ 18 \end{bmatrix}, Eqin = \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \end{bmatrix}
$$

Initially, we begin by searching for inequality constraints that contain only one nonzero element. We observe that, in the first equality constraint, all elements are zero except for the third element:

$$
-3x_3 \leq 2
$$

Following the first second case of the previous subsection, the first row is redundant and can be deleted. So we can update matrix A and the first element from vectors b and $Eqin$:

$$
A = \begin{bmatrix} 4 & -3 & 8 & -1 \\ -3 & 2 & 8 & -4 \\ 0 & 0 & -1 & 0 \end{bmatrix}, c = \begin{bmatrix} -2 \\ 4 \\ -2 \\ 2 \end{bmatrix}, b = \begin{bmatrix} 20 \\ -8 \\ 18 \end{bmatrix}, Eqin = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
$$

Finally, the presolved problem is:

$$
\begin{aligned}
\min \quad & z = -2x_1 + 4x_2 - 2x_3 + 2x_4 \\
\text{s.t.} \quad & -4x_1 - 3x_2 + 8x_3 - x_4 = 20 \\
& -3x_1 + 2x_2 - 4x_4 \geq -8 \\
& - x_3 = 18 \\
& x_j \geq 0, \qquad (j = 1, 2, 3, 4)
\end{aligned}
$$

3.2.5. Eliminate dual singleton inequality constraints.

This method is similar to the previous one but is applied to the dual LP problem. A column in the coefficient matrix A is considered a singleton column if and only if it contains exactly one nonzero coefficient. When transforming the primal LP problem to its dual, a singleton column in the primal corresponds to a dual singleton constraint in the dual LP problem. A dual singleton inequality row can be formulated as follows:

$$A_{j1}w_1 + A_{j2}w_2 + \cdots + A_{jm}w_m \leq c_j$$

Where $A_{jk} \neq 0 \wedge A_{ji} = 0, i = 1, 2, \dots, m,\ j = 1,2, \dots, n,\ and\ i \neq k$. A constraint of this type may either be redundant or indicate that the dual LP problem is unbounded and the primal LP problem is infeasible. We present this method and the corresponding eliminations without needing to transform the initial problem to its dual. Instead of eliminating a row in the dual LP problem, we eliminate a column in the primal LP problem. The following theorem distinguished all the possible cases.

For each dual singleton inequality constraint, we distinguish the following cases:

1. Constraint type $\leq, A_{kj} > 0 \ and \ c_j > 0$: Column j is redundant and can be deleted.
2. Constraint type $\leq, A_{kj} < 0 \ and \ c_j < 0$: The LP problem is infeasible.
3. Constraint type $\leq, A_{ik} > 0 \ and \ c_j = 0$: Column j is redundant and can be deleted.
4. Constraint type $\leq, A_{ik} < 0 \ and \ c_j = 0$: Row k and column j are redundant and can be deleted.
5. Constraint type $\geq, A_{ik} > 0 \ and \ c_j < 0$: The LP problem is infeasible.
6. Constraint type $\geq, A_{ik} < 0 \ and \ c_j > 0$: Column j is redundant and can be deleted.
7. Constraint type $\geq, A_{ik} > 0 \ and \ c_j = 0$: Row k and column j are redundant and can be deleted.
8. Constraint type $\geq, A_{ik} < 0 \ and \ c_j = 0$: Column j is redundant and can be deleted.

Throughout the next illustrative example this method is demonstrated.

$$\begin{aligned} \min \quad & z = 4x_1 + x_2 - 2x_3 + 7x_4 \\ \text{s.t.} \quad & 3x_1 - x_3 - 6x_4 \leq 0 \\ & -3x_1 + 2x_2 + 5x_3 - x_4 \geq 0 \\ & 4x_1 + 3x_3 + 4x_4 \leq 5 \\ & x_j \geq 0, \quad (j = 1, 2, 3, 4) \end{aligned}$$

In matrix notation:

$$A = \begin{bmatrix} 3 & 0 & -1 & -6 \\ -3 & -2 & 5 & -1 \\ 4 & 0 & 3 & 4 \end{bmatrix}, c = \begin{bmatrix} 4 \\ 1 \\ -2 \\ 7 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}, Eqin = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

Initially, we begin by searching for columns that contain only one nonzero element. We observe that, in the second column, all elements are zero except for the third element. According to the sixth case of the theorem, the second column is redundant and can be deleted. We can delete the second column of matrix A and the second element from vector c:

$$A = \begin{bmatrix} 3 & -1 & -6 \\ -3 & 5 & -1 \\ 4 & 3 & 4 \end{bmatrix}, c = \begin{bmatrix} 4 \\ -2 \\ 7 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}, Eqin = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

Finally, the presolved LP problem is:

$$\begin{aligned} \min \quad & z = 4x_1 - 2x_3 + 7x_4 \\ \text{s.t.} \quad & 3x_1 - x_3 - 6x_4 \leq 0 \\ & -3x_1 + 5x_3 - x_4 \geq 0 \\ & 4x_1 + 3x_3 + 4x_4 \leq 5 \\ & x_j \geq 0, \quad (j = 1, 2, 3, 4) \end{aligned}$$

3.2.6. Eliminate implied free singleton columns.

A constraint that implies a free singleton column can be formulated as:

$$A_{i.}x + A_{is}x_s = b$$

where $i = 1, 2, \dots, m$, and the singleton column inside it ($A_{is} \neq 0$) is redundant if and only if:

$$A_{is} > 0 \wedge A_{ij} \leq 0, j \neq s, j = 1,2, \dots, n$$

or:

$$A_{is} < 0 \wedge A_{ij} \geq 0, j \neq s, j = 1,2, \dots, n$$

In this scenario, we can remove variable x_s from the LP problem. Furthermore, we have the option to eliminate constraint i. If $c_s = 0$, we delete only constraint i. If $c_s \neq 0$, we update vector c and adjust the constant term of the objective function (c_0):

$$c = c - \frac{c_s}{A_{is}} A_{i.}^T$$

$$c_0 = c_0 + \frac{c_s}{A_{is}} b_i$$

Throughout the next illustrative example this method is demonstrated.

$$\begin{aligned} \min \quad & z = x_1 + 2x_2 - 4x_3 - 3x_4 \\ \text{s.t.} \quad & 3x_1 + 5x_3 + 2x_4 \leq 5 \\ & -x_1 + 2x_2 - 3x_4 = 8 \\ & -2x_1 - 2x_3 + x_4 \geq 6 \\ & x_j \geq 0, \quad (j = 1, 2, 3, 4) \end{aligned}$$

In matrix notation:

$$A = \begin{bmatrix} 3 & 0 & 5 & 2 \\ -1 & 1 & 0 & -3 \\ -2 & 0 & -2 & 1 \end{bmatrix}, c = \begin{bmatrix} 1 \\ 2 \\ -4 \\ -3 \end{bmatrix}, b = \begin{bmatrix} 5 \\ 8 \\ 6 \end{bmatrix}, Eqin = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Initially, we begin by searching for columns that contain only one nonzero element. We observe that, in the second column, all elements are zero except for the second element. According to the first case, the second column is redundant and can be deleted. First, we update vector c and calculate the constant term of the objective function, assuming that its initial value is zero:

$$c_0 = c_0 + \frac{c_2}{A_{22}} b_2 = 0 + \frac{2}{1} 8 = 16 c = c - \frac{c_2}{A_{22}} A_{2.}^T = \begin{bmatrix} 1 \\ 2 \\ -4 \\ -3 \end{bmatrix} - \frac{2}{1} \begin{bmatrix} -1 \\ 1 \\ 0 \\ -3 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ -4 \\ -3 \end{bmatrix}$$

So, we delete the second row and the second column for matrix A and the second element from vectors c, b and Eqin:

$$A = \begin{bmatrix} 3 & 5 & 2 \\ -2 & -2 & 1 \end{bmatrix}, c = \begin{bmatrix} 3 \\ -4 \\ -3 \end{bmatrix}, b = \begin{bmatrix} 5 \\ 6 \end{bmatrix}, Eqin = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, c_0 = 16$$

So, the presolved LP problem is:

$$\begin{aligned} \min \quad & z = x_1 - 4x_3 - 3x_4 + 16 \\ \text{s.t.} \quad & 3x_1 + 5x_3 + 2x_4 \leq 5 \\ & -2x_1 - 2x_3 + x_4 \geq 6 \\ & x_j \geq 0, \quad (j = 1, 3, 4) \end{aligned}$$

3.2.7. Eliminate redundant columns.

A linear constraint of the form:

$$A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{ik}x_k = 0$$

With all $A_{ij} > 0, i = 1, 2, \ldots, m,\ j = 1,2, \ldots, k,\ 1 \leq k \leq n, implies\ that\ x_j = 0$

Also, a linear constraint of the form:

$$A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{ik}x_k = 0$$

With all $A_{ij} < 0, i = 1, 2, \ldots, m,\ j = 1,2, \ldots, k,\ 1 \leq k \leq n, implies\ that\ x_j = 0$

In both cases, all variables $x_j = 0, j = 1,2, \ldots, k,\ 1 \leq k \leq n,$ are redundant an can be deleted. Consequently, the constraints shown above are also linearly dependent and they can be deleted.

With this example, the presolved method that eliminates redundant columns is demonstrated. The LP problem is the following:

$$\begin{aligned}
\text{min} \quad & z = -6x_1 + 4x_2 - 2x_3 - 8x_4 \\
\text{s.t} \quad & 2x_1 - 4x_2 + 2x_3 + 2x_4 = 20 \\
& -6x_2 - 2x_3 + 2x_4 + x_5 = 26 \\
& 2x_2 + 8x_3 + x_6 = 0 \\
& 16x_2 - 12x_3 - 8x_4 = -84 \\
x_j \geq 0, \quad & (j = 1, 2, 3, 4, 5, 6)
\end{aligned}$$

In matrix notation:

$$A = \begin{bmatrix} 2 & -4 & 2 & 2 & 0 & 0 \\ 0 & -6 & -2 & 2 & 1 & 0 \\ 0 & 2 & 8 & 0 & 0 & 1 \\ 0 & 16 & -12 & -8 & 0 & 0 \end{bmatrix}, c = \begin{bmatrix} -6 \\ 4 \\ -2 \\ -8 \\ 0 \\ 0 \end{bmatrix}, b = \begin{bmatrix} 20 \\ 26 \\ 0 \\ -84 \end{bmatrix}, Eqin = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Initially, we begin by searching for equality constraints with a zero in the right hand side. We observe that, in the third constraint, the right hand is zero. Also, all the elements in the third row are greater or equal to zero.

Variables x_2, x_3 and x_6 are redundant and can be deleted:

$$A = \begin{bmatrix} 2 & 2 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 0 \\ 0 & -8 & 0 \end{bmatrix}, c = \begin{bmatrix} -6 \\ -8 \\ 0 \end{bmatrix}, b = \begin{bmatrix} 20 \\ 26 \\ 0 \\ -84 \end{bmatrix}, Eqin = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Also, the third constraint and can be eliminated too. The presolved LP problem is:

$$\begin{aligned} \min \quad & z = -6x_1 - 8x_4 \\ \text{s.t.} \quad & 2x_1 + 2x_4 = 20 \\ & + 2x_4 + x_5 = 26 \\ & + 2x_4 = -84 \\ & x_j \geq 0, \qquad (j = 1, 4, 5) \end{aligned}$$

3.2.8. Eliminate implied bounds on rows.

A constraint that implies new bounds for the constraints can be formulated as:

$$\underline{b_i} \leq A_{i.}x \leq \overline{b_i}$$

and:

$$\underline{x} \leq x \leq \overline{x}$$

where $\underline{x} = 0$ and $\overline{x} = +\infty$. These new bounds can be computed by:

$$\underline{b_i}' = inf_{\underline{x} \le x \le \overline{x}} A_{i.}x = \sum_{A_{ij} \ge 0} A_{ij}\underline{x_j} + \sum_{A_{ij} \le 0} A_{ij}\overline{x_j}$$

$$\overline{b_i}' = sup_{\underline{x} \le x \le \overline{x}} A_{i.}x = \sum_{A_{ij} \ge 0} A_{ij}\overline{x_j} + \sum_{A_{ij} \le 0} A_{ij}\underline{x_j}$$

These equations calculate both the greatest from the inferior bounds and the smallest from the superior bounds. If $[\underline{b_i}', \overline{b_i}'] \cap [\underline{b_i}, \overline{b_i}] = \emptyset$, then the LP problem is infeasible. If $[\underline{b_i}', \overline{b_i}']\subset [\underline{b_i}, \overline{b_i}]$, then the constraint i is redundant and can be deleted. All the possible cases are distinguished in the following theorem.

For each constraint that implies new bounds, we distinguished the following cases:

1. Constraint type $\le, A_{i.} \le 0 \ \ and \ \overline{b_i} \ge 0$: Row i is redundant and can be deleted.
2. Constraint type $\le, A_{kj} \ge 0 \ \ and \ \underline{b_i} \le 0$: Row i is redundant and can be deleted.

With this example, the presolved method that eliminates redundant columns is demonstrated. The LP problem is the following:

$$\begin{aligned}
\text{min} \quad z = \ & 3x_1 - 4x_2 + 5x_3 - 2x_4 \\
\text{s.t} \quad & -2x_1 - x_2 - 4x_3 - 2x_4 \le 4 \\
& 5x_1 + 3x_2 + x_3 + 2x_4 \le 18 \\
& 5x_1 + 3x_2 + x_4 \ge -13 \\
& 4x_1 + 6x_2 + 2x_3 + 5x_4 \ge -10 \\
x_j \ge 0, \quad & (j = 1, 2, 3, 4)
\end{aligned}$$

In matrix notation:

$$A = \begin{bmatrix} -2 & -1 & -4 & -2 \\ 5 & 3 & 1 & 2 \\ 5 & 3 & 0 & 1 \\ 4 & 6 & 2 & 5 \end{bmatrix}, c = \begin{bmatrix} 3 \\ -4 \\ 5 \\ -2 \end{bmatrix}, b = \begin{bmatrix} 4 \\ 18 \\ -13 \\ -10 \end{bmatrix}, Eqin = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

Initially, we begin by searching for inequality constraints, in this case all of them are. Looking at the first case of the theorem, the first row and the third row are redundant and can be deleted. Also, looking at the second case of the theorem, the fourth row is redundant and can be deleted:

$$A = [5 \quad 3 \quad 1 \quad 2], c = \begin{bmatrix} 3 \\ -4 \\ 5 \\ -2 \end{bmatrix}, b = [18], Eqin = [-1]$$

Finally, the presolved problem is:

$$\begin{aligned} \min \quad & z = 3x_1 - 4x_2 + 5x_3 - 2x_4 \\ \text{s.t.} \quad & 5x_1 + 3x_2 + x_3 + 2x_4 \leq 18 \\ & x_j \geq 0, \qquad (j = 1, 2, 3, 4) \end{aligned}$$

3.2.9. Eliminate redundant rows.

Two constraints i and k are linearly dependent if and only if $A_i = \lambda A_k$. The identification of linearly dependent constraints can be done by calculating the rank

of the coefficient matrix A using the augmented matrix. $[A|b]$ and performing adequate row operations until the identity matrix is derived. The LP problem must be in standard form. The redundant constraints that must be deleted are:

$$A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n = 0$$

Where $A_{ij} = 0, i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

If the constraint looks like the next one, the problem is infeasible.

$$A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n = b_i$$

Where $A_{ij} = 0, b_i \neq 0, i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

The next example portraits this method.

$$\begin{aligned}
\text{min} \quad z = \quad & x_1 + x_2 - 2x_3 - 3x_4 \\
\text{s.t} \quad & x_1 + 2x_2 + 4x_3 + 5x_4 = 10 \\
& 3x_1 + 3x_2 + 8x_3 + 4x_4 = 2 \\
& 0.5x_1 + x_2 + 2x_3 + 2.5x_4 = 5 \\
x_j \geq 0, \quad & (j = 1, 2, 3, 4)
\end{aligned}$$

In matrix notation:

$$A = \begin{bmatrix} 1 & 2 & 4 & 5 \\ 3 & 5 & 8 & 4 \\ 0.5 & 1 & 2 & 2.5 \end{bmatrix}, c = \begin{bmatrix} 1 \\ 1 \\ -2 \\ -3 \end{bmatrix}, b = \begin{bmatrix} 10 \\ 2 \\ 5 \end{bmatrix}, Eqin = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The augmented matrix $[A|b]$:

$$Augmented = \begin{bmatrix} 1 & 2 & 4 & 5 & \vdots & 10 \\ 3 & 5 & 8 & 4 & \vdots & 2 \\ 0.5 & 1 & 2 & 2.5 & \vdots & 5 \end{bmatrix}$$

Looking at the first and third row, they are linearly dependent, because the third is the first one divided by two. Because of this we can deleted one of them, having the equivalent LP problem after presolve:

$$\begin{aligned} \text{min} \quad z = \quad & x_1 + x_2 - 2x_3 - 3x_4 \\ \text{s.t} \quad & x_1 + 2x_2 + 4x_3 + 5x_4 = 10 \\ & 3x_1 + 3x_2 + 8x_3 + 4x_4 = 2 \\ x_j \geq 0, \quad & (j = 1, 2, 3, 4) \end{aligned}$$

4. Problems battery under study

The battery of problems that is going to be study throughout the project is chosen from the model library of the software GAMS Studio. Thirty LP problems from the entire library are selected, solved and simplified with the *sparsification* method, to be later analysed. The battery covers a wide range of different topics.

4.1. Agricultural Economics

- AGRESTE: Agricultural farm level model of NE Brazil.
- CHINA: Organic fertilizer use in intensive farming.
- DEMO1: Simple farm level model.
- PAKLIVE: Pakistan Punjab livestock model.

4.2. Management Science and OR

- AIRCRAFT: Aircraft allocation under uncertain demand.
- AMPL: AMPL sample problem.
- DECOMP: Decomposition principle.
- GUSSEX1: simple GUSS example.
- GUSSGRID: simple GUSS grid example
- IBM1: Aluminium alloy smelter sample problem.
- JOBT: On-the-job training.
- MINE: Opencast Mining.
- PRODMIX: A production mix problem.
- ROBERT: Elementary production and inventory model.
- SENSTRAN: Sensitivity analysis using loops.
- SPARTA: Military manpower planning from wanger.
- TRNSPORT: A transportation problem.
- UIMP: Production scheduling problem.
- WHOUSE: Simple warehouse problem.

4.3. Stochastic Programming

- AIRSP: Aircraft allocation.

- CLEARLAK: Scenario reduction: Clearklake exercise.
- MARKOV: Strategic petroleum reserve.
- SRKANDW: Stochastic programming scenario reduction.
- SRPCHASE: Scenario tree construction example.
- KAND: Stochastic problem.
- LANDS: Optimal investment.

4.4. Macro and Micro economics

- DIET: Stigler's nutrition model.
- MEXSS: Mexico Steel – Small Static.
- ORANI: A multisector price endogenous model of Australia.

4.5. Mathematics

- DEA: Data envelopment analysis.
- IMSL: Piecewise Linear approximation.
- QP5: Standard QP model – linear approximation.

5. Sensibility analysis

The sensibility analysis aims to portrait and evaluate graphically the evolution of the battery of problems, for different levels of simplification (epsilon). The analysis will focus on how the objective function reacts to the operation, while the unfeasibility of the problem is controlled too. The entire analysis is program with Python in Jupyter Notebook.

60 linear programming problems are solved and simplified with the same algorithm, minimizing the objective function. The simplification is done for thirty values of epsilon and the results obtained are organised on vectors of length thirty (one for each epsilon). These results are saved into a json file, with the 60 probemes, and 6 keys for each one. The keys considered are:

- Epsilon: signification level that is used to modify the matrix A, if the element is less than epsilon (matrix A normalized) it takes 0 value.
- Objective_function: With the new matrix A (modified using epsilon), a new model is built. This is the value for the new objective function for each epsilon.
- Decision_variables: It contains the value of each decision variable for each epsilon in the new model built with the modified matrix A.
- Changed_indices: Elements that have been modified from the matrix A, for each value of epsilon.
- Constraint_violation: Taking the solution of the new model (modified matrix A) and putting it in the original model, the violation of each constraint is evaluated, for each epsilon value.
- of_original_decision: Taking the solution of the new model (modified matrix A) and putting it in the original model, the objective function is evaluated, for each epsilon value.
- time_required: It shows the time required to solved the problem for each epsilon value.

The json file is read directly in Jupyter notebook and all the data is used here. For the analysis three main indexes for each epsilon and for each problem are calculated: a normalized objective function, an unfeasibility index and a problem complexity index.

Firstly, the study is done for a small sample of problems, and afterwards a function is made to iterate the complete battery.

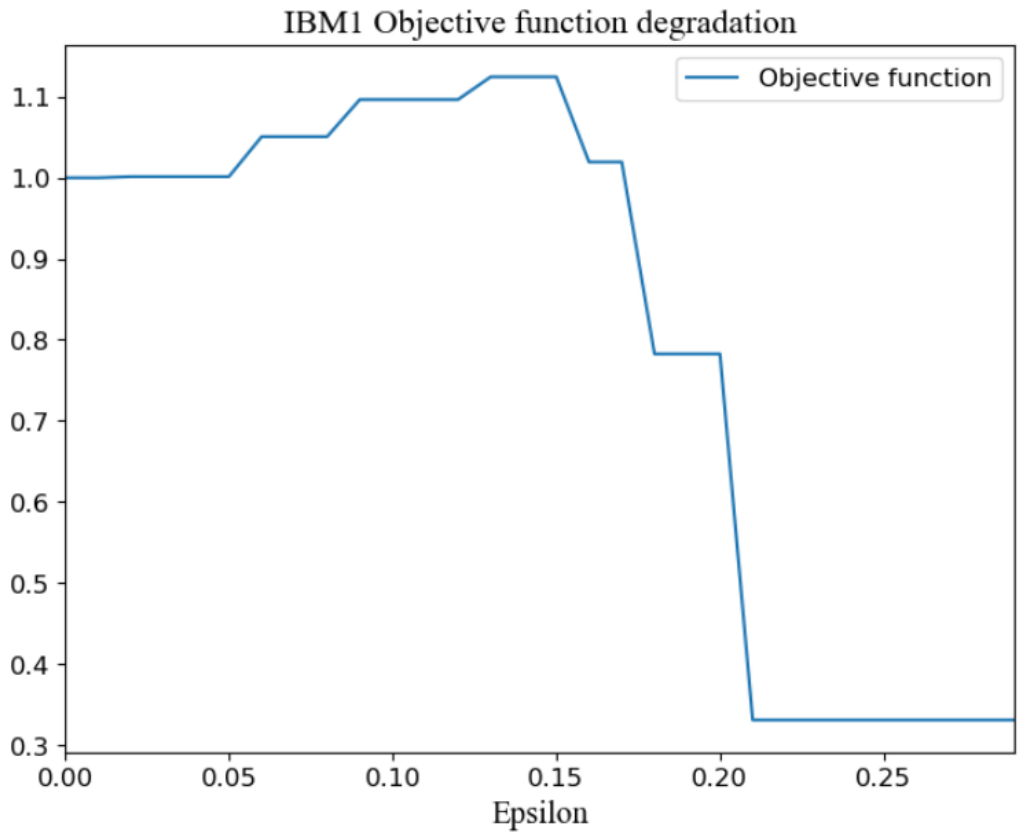
5.1. Indexes

The first index calculated is the normalized objective function. In order to reach it, the objective function value for each epsilon is divided by the first one. After iterating all the epsilons, there is a vector with thirty elements, one for each epsilon, with the normalised objective function. The graph of this vector will show the degradation of the objective function while the simplification increases. (Codes of the operations needed to obtain the indexes in the annex). The equation to obtain this index is:

$$Ob\ func\ pu\ = \frac{Ob\ func}{Ob\ func[0]}$$

Being *Ob_func_pu* the list with the normalized objective function values, *Ob_func* the list with the objective function values and *Ob_func[0]* the optimal solution.

Example:



1. Index that measures the objective function degradation.

Secondly, the unfeasibility index is calculated. Before making any operation, it is applied a filter, to only consider de violations bigger than 10^{-6}, to avoid operational mistakes. Because not all the constraints have the same weight in the objective function, it is necessary to give to each constraint violation its weight. This is done by multiplying them by the dual value of each restriction. Afterwards, all the absolute values of the weighted constraints violations are summed, getting a value for each epsilon. These values depend on the data of the problem, so they need to be normalized, therefore the actual index is divided by the optimal solution for each problem. Iterating all the epsilons, the solution is a vector with thirty elements, one for one epsilon, measuring how infeasible the problem is. The graph of this vector shows the evolution of the unfeasibility of the problem while the simplification increases. The equations to obtain this index are:

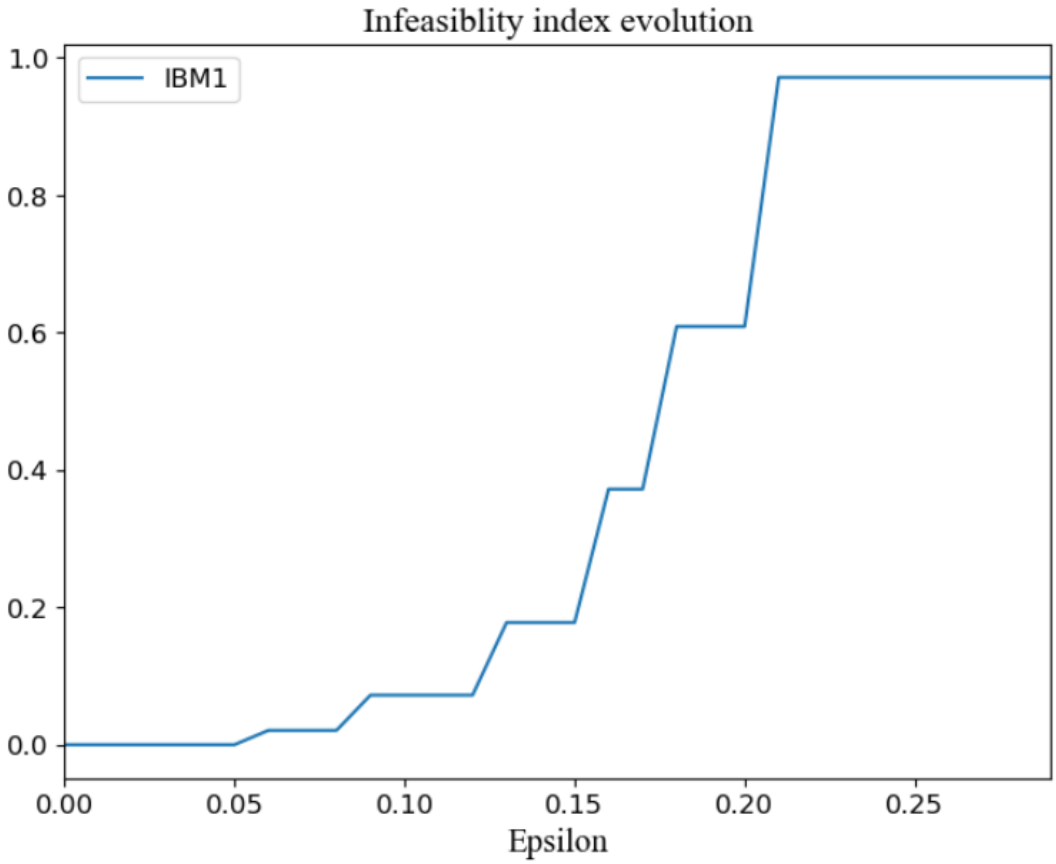
$$product\_cv\_vd = const\_violations \times var\_dual$$

Being const_violations the list of constraint violations for each value of epsilon, and var_dual the list of dual decision variables for each value of epsilon. product_cv_vd is a list of lists, so in order to have one value for each epsilon, the sublists for every epsilons are sum together, with coding, and is saved into a new variable called sum. Once the variable sum has a value for each epsilon the next equation is used:

$$unfeasibility\ index = \frac{sum}{Ob\ func[0]}$$

Being *Ob_func[0]* the optimal solution.

Example:



2. Index that measures the problem infeasibility.

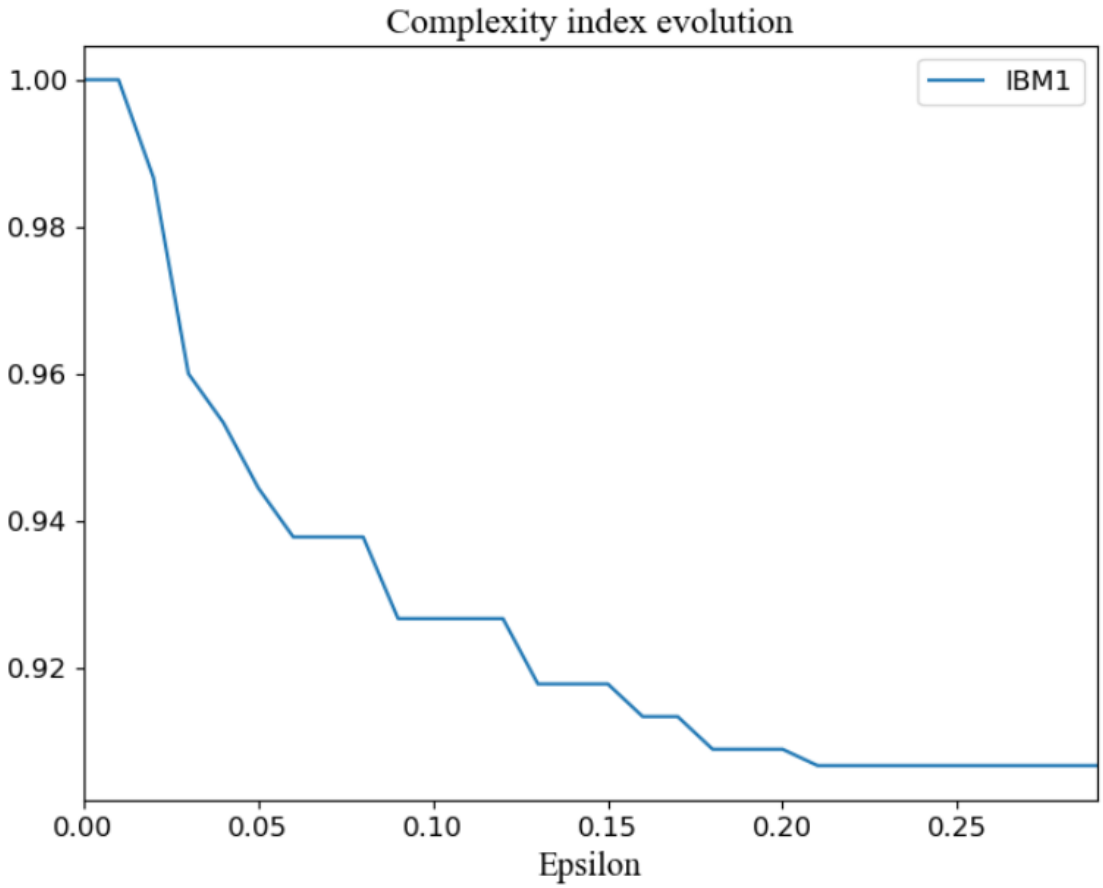
Lastly, to see the evolution of the complexity while the simplification goes on, the complexity problem index is calculated. First of all, the total number of elements in the matrix A is obtained by multiplying the number of variables by the number of constraints. This value will be used to normalize the index, being the index always positive and between zero and one. In the key changed_indices there is the number of elements of the matrix that are zero for each epsilon, so after dividing this number by the total number of elements, by getting the opposite (one minus the value before) the complexity index is obtained. The equations to obtain the index are:

$$mat\ A\ elem\ = dec\ var * dual\ dec\ var$$

$$complexity\ index = 1 - \frac{elements\ 0}{mat\ A\ elem}$$

Being *elements 0* the list with matrix A elements that are 0 for each epsilon

Example:



3. Index that measures the problem complexity.

5.2. Procedure of the analysis

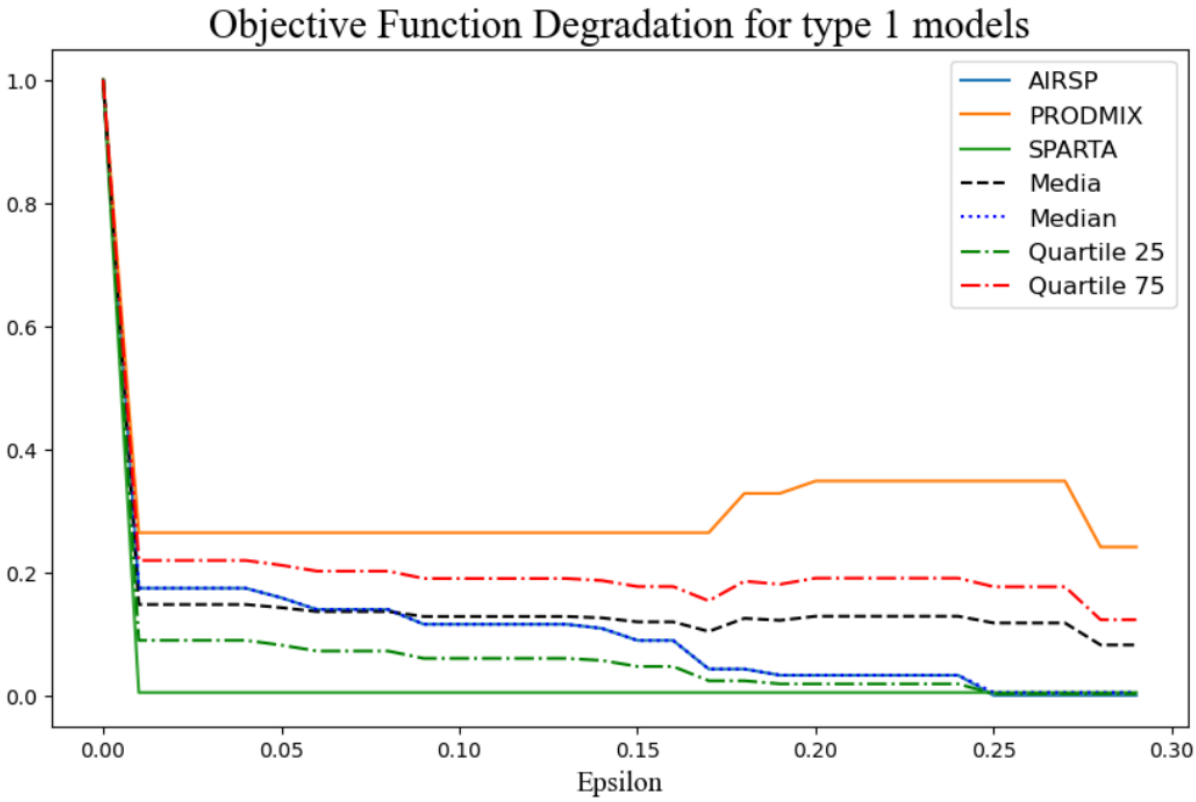
Once that all the operations needed to obtain the indexes are done, the next part is to integrate them into a big function, to be able to iterate the entire battery of problems with a loop. The analysis is based on three graphs for each problem showing the three indexes evolution throughout the simplification operation. By iterating all the problem all the graphs are obtain and the analysis can begin.

The analysis consists of looking for the different answers that each problem gives to the simplification, in order to find similarities or a pattern in some of the graphs. Before the analysis begin, it is expected that the objective function takes a close value to the optimum, while the complexity decreases, and the infeasibility

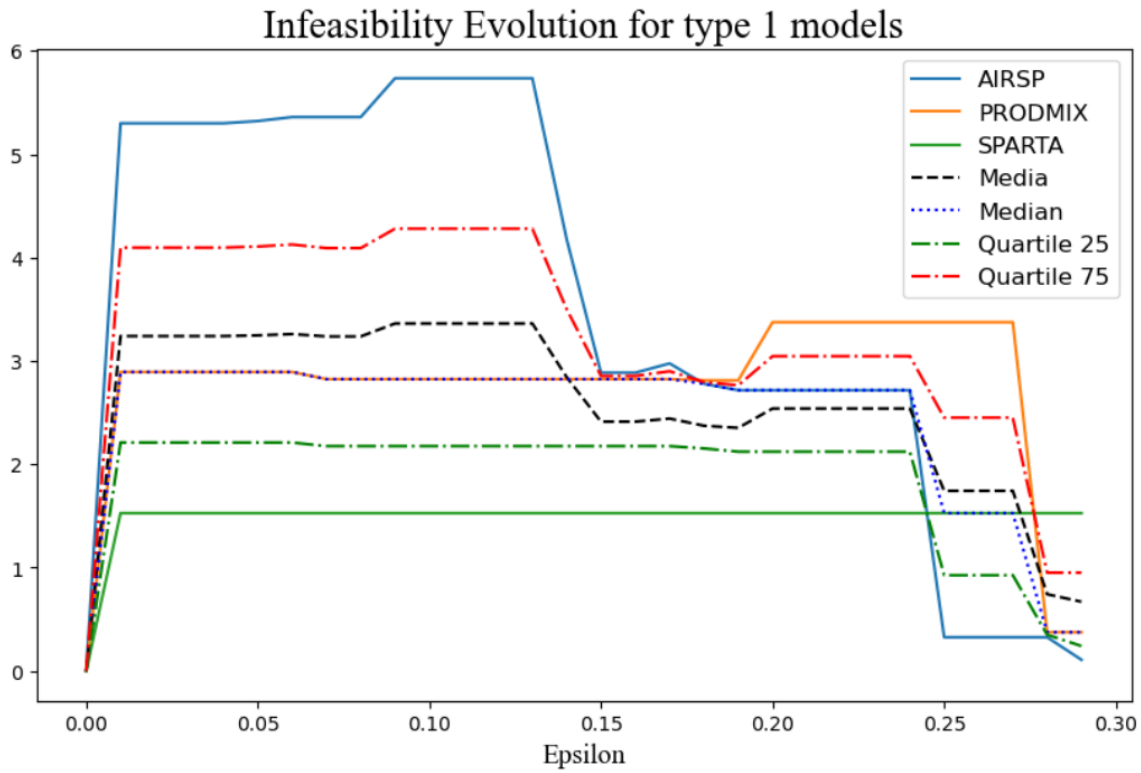
worsens in a barely noticeable manner. Taking a close look at the graphs, we can observe four main behaviours in response to the simplification. For each one three graphs with the indexes are obtained. Additionally, in this graphs statistics of media, median and quartiles (25 and 75) are included.

The first behaviour appreciated is the problems that the objective function improves in a great percentage, but they become notably infeasible in a very swift manner. For these type of problems, the simplification does not have sense, and the optimal solution that was reached in the beginning is the best one. The graph only shows 3 models as representatives of this behaviour for clarity reasons, but more than 15 had this behaviour.

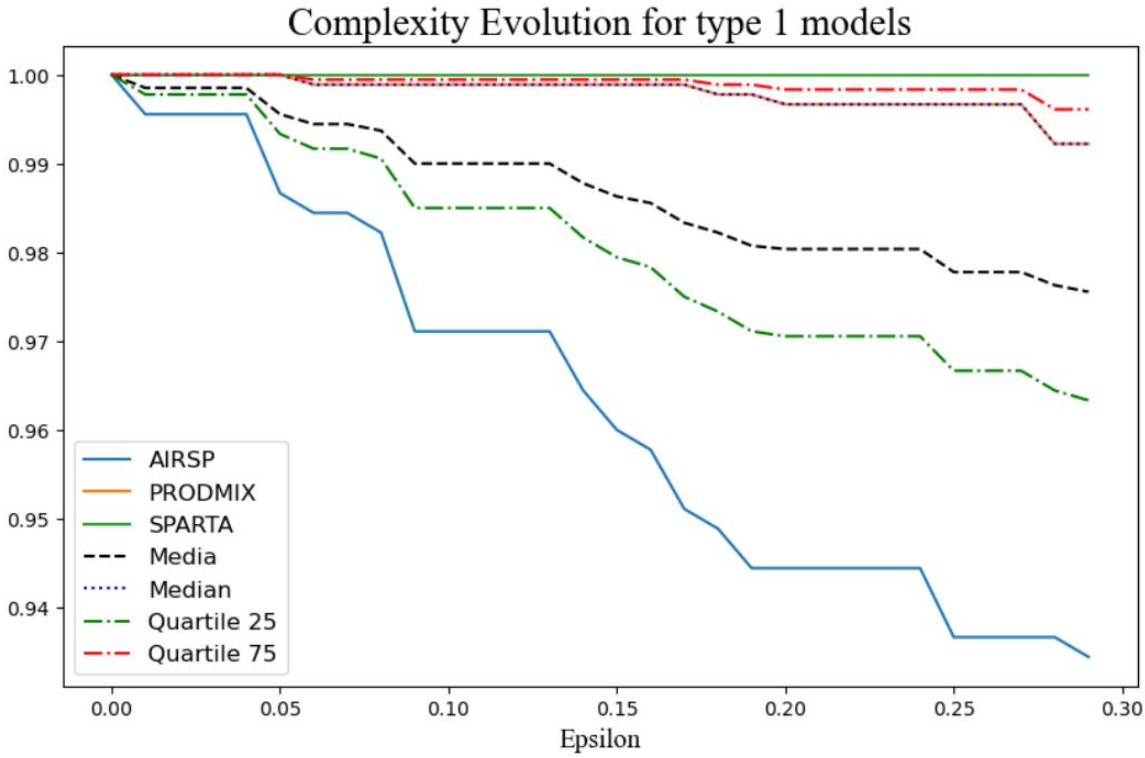
Example:



4. Objective function evolution from type 1 problems.



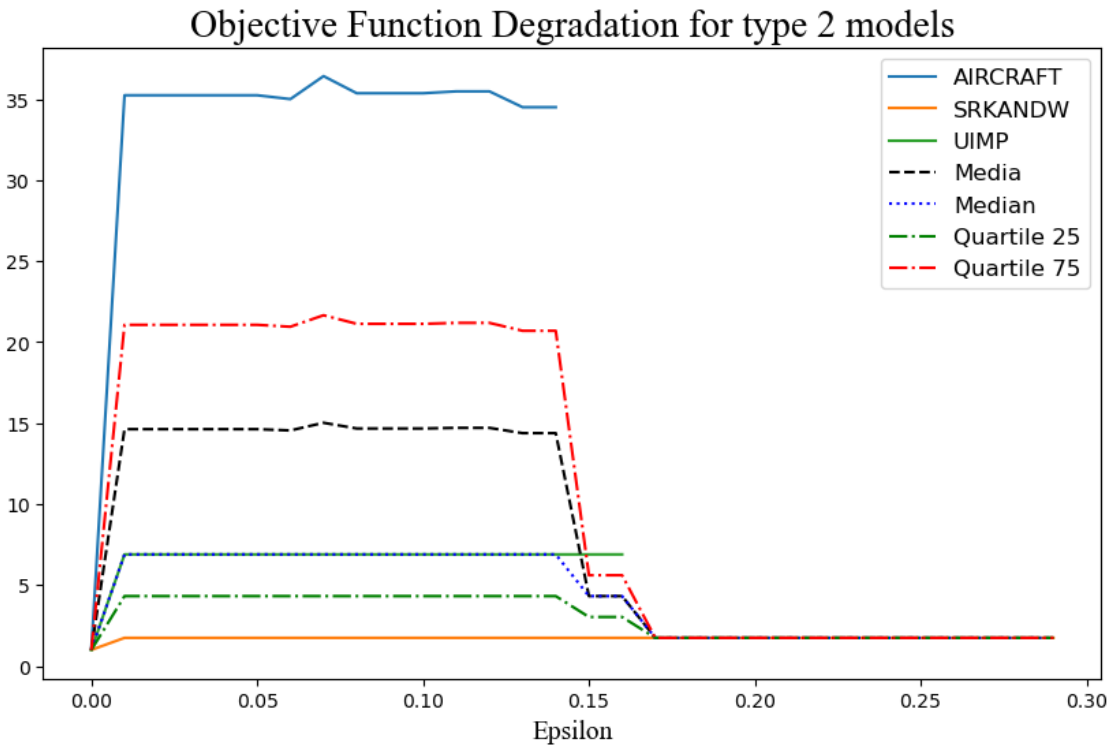
5. Infeasibility evolution from type 1 problems.



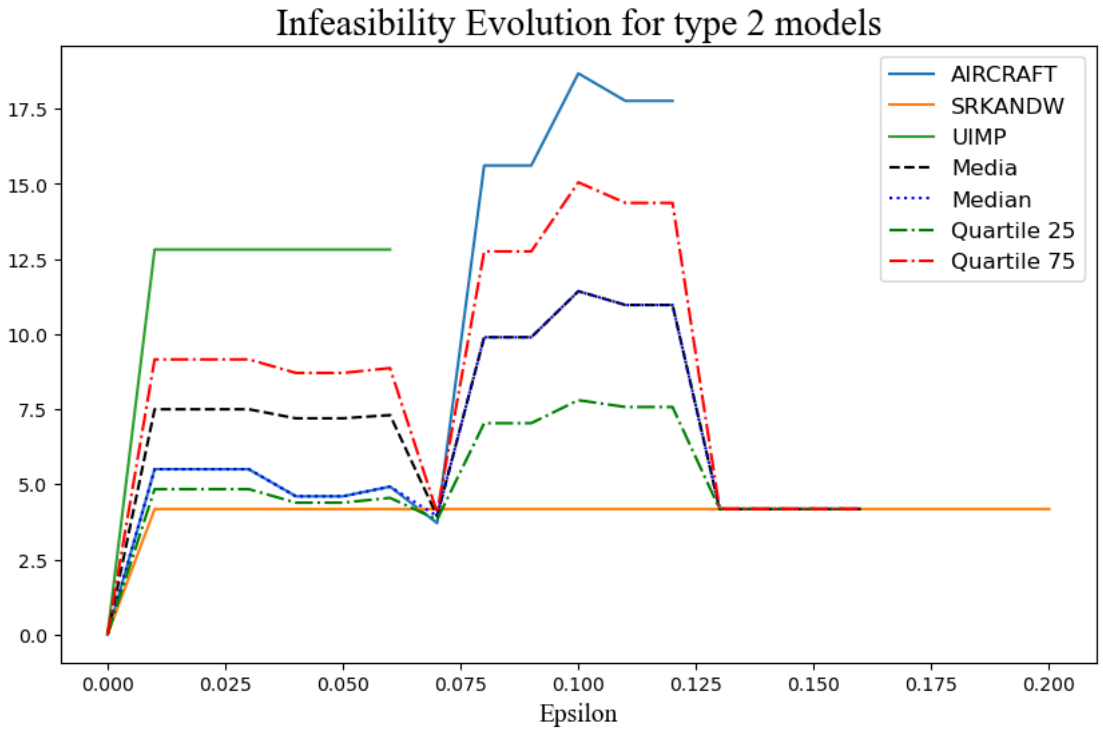
6. Complexity evolution from type 1 problems.

The next behaviour is quite similar to the one before. This time the objective function worsens, but the problems still become notably infeasible very quickly. Again, the simplification for these models does not have any sense. As in the other two, the graph does not show all the models that follow this behaviour, but just some of them, so the graph is clearer.

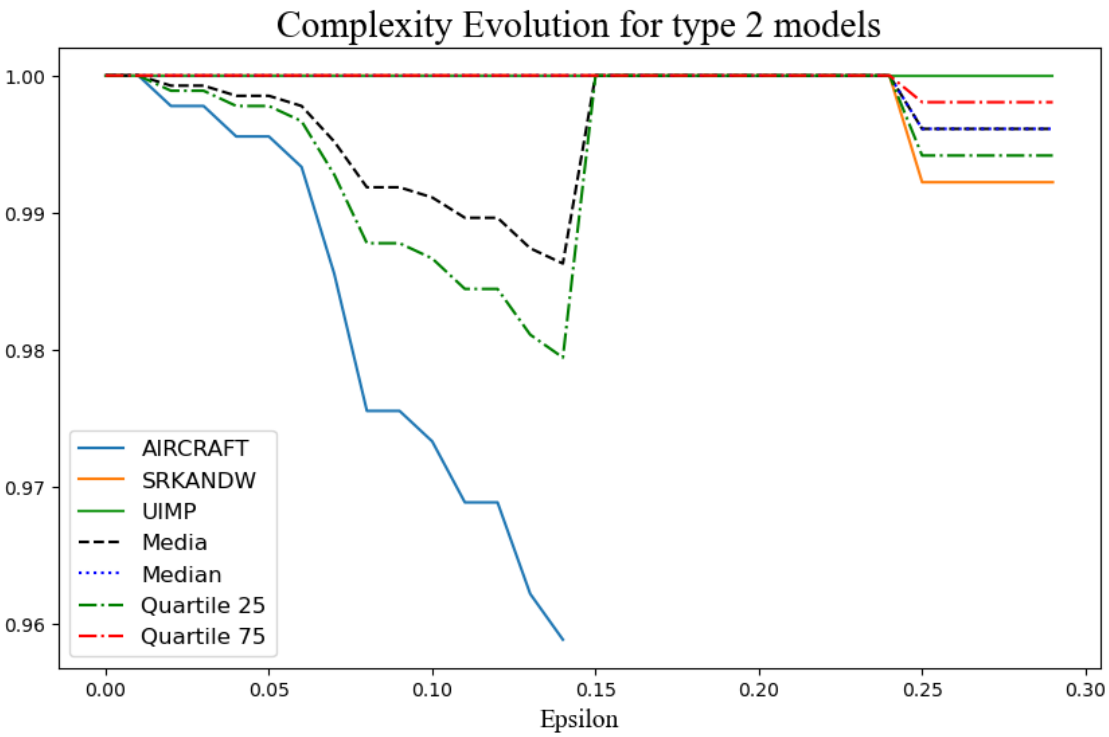
Example:



Objective functions from type 2 problems.



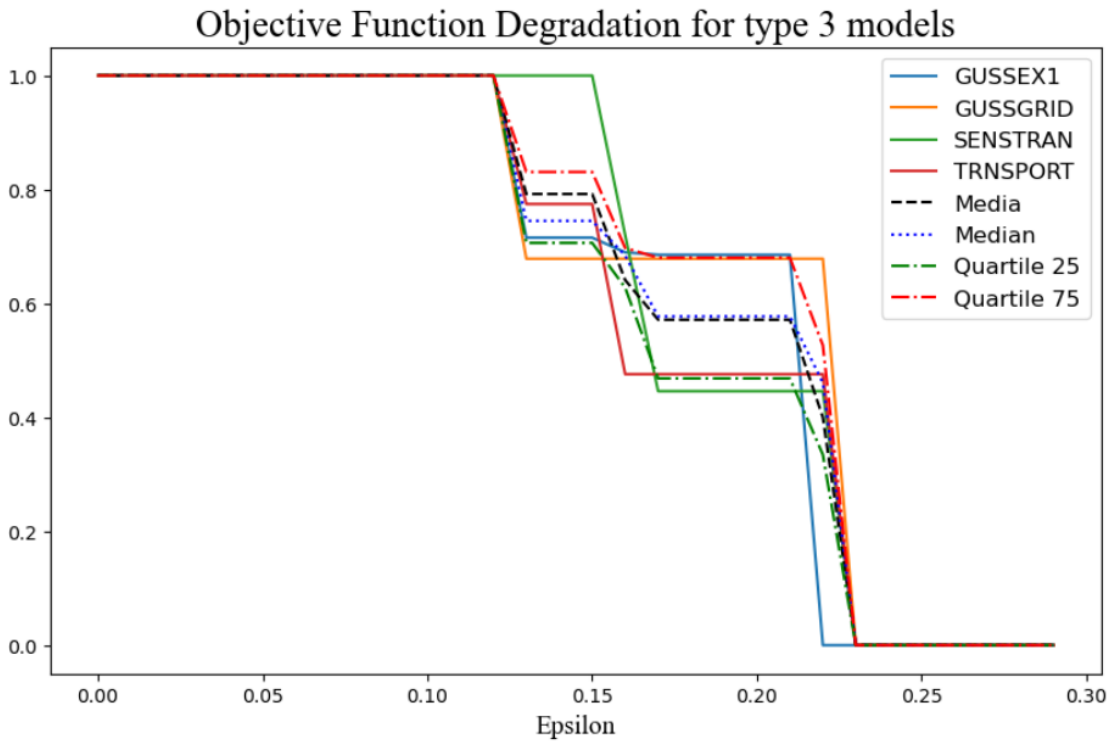
Infeasibility from type 2 problems.



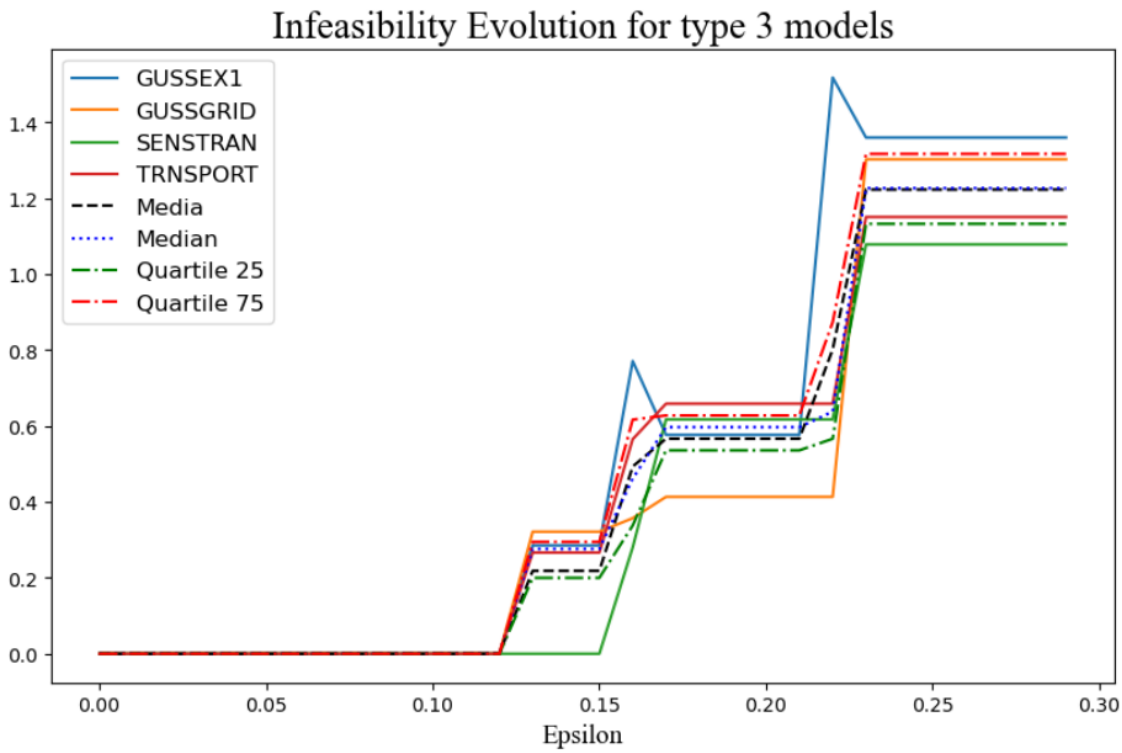
Complexity from type 2 problems.

The third behaviour that appeared is the problems which does not see themselves affected by the simplification until big epsilons. When this happens, the objective function improves at the same rate that the problem becomes infeasible. It is not until this time that the complexity of the problem decreases. So again, the simplification of this problems is not possible. Again, here for clarity reasons not all the models that follows this behaviour are included in the graph.

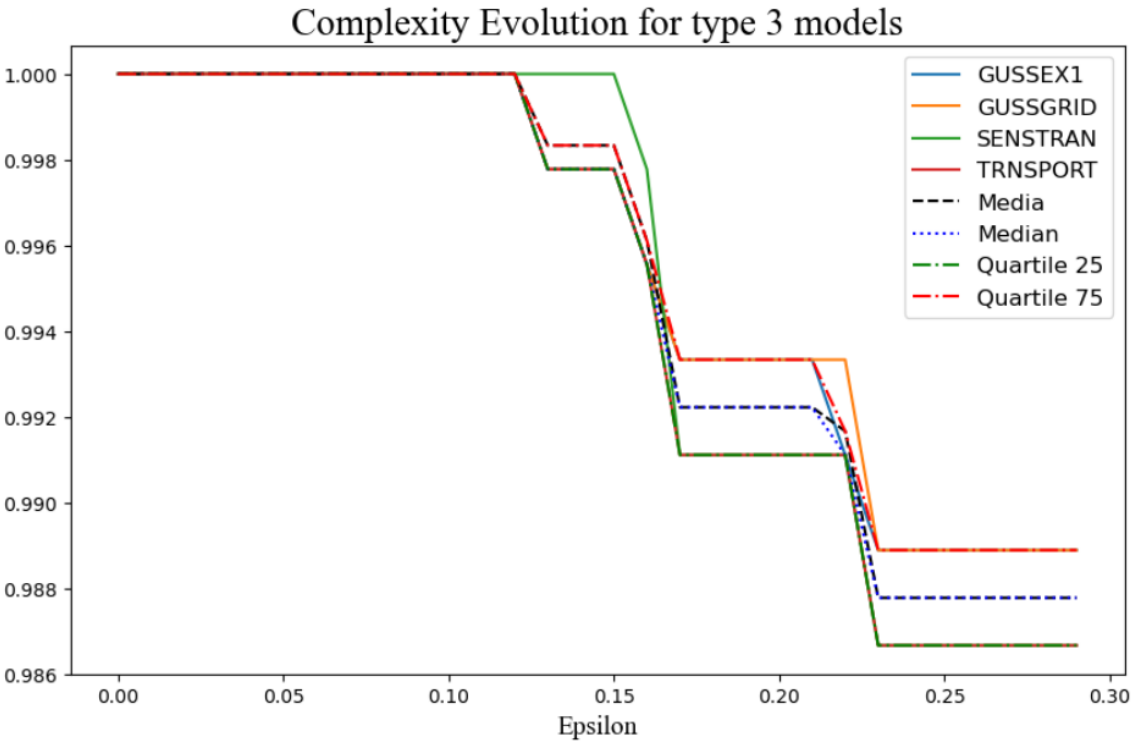
Example:



Objective functions from type 3 problems.

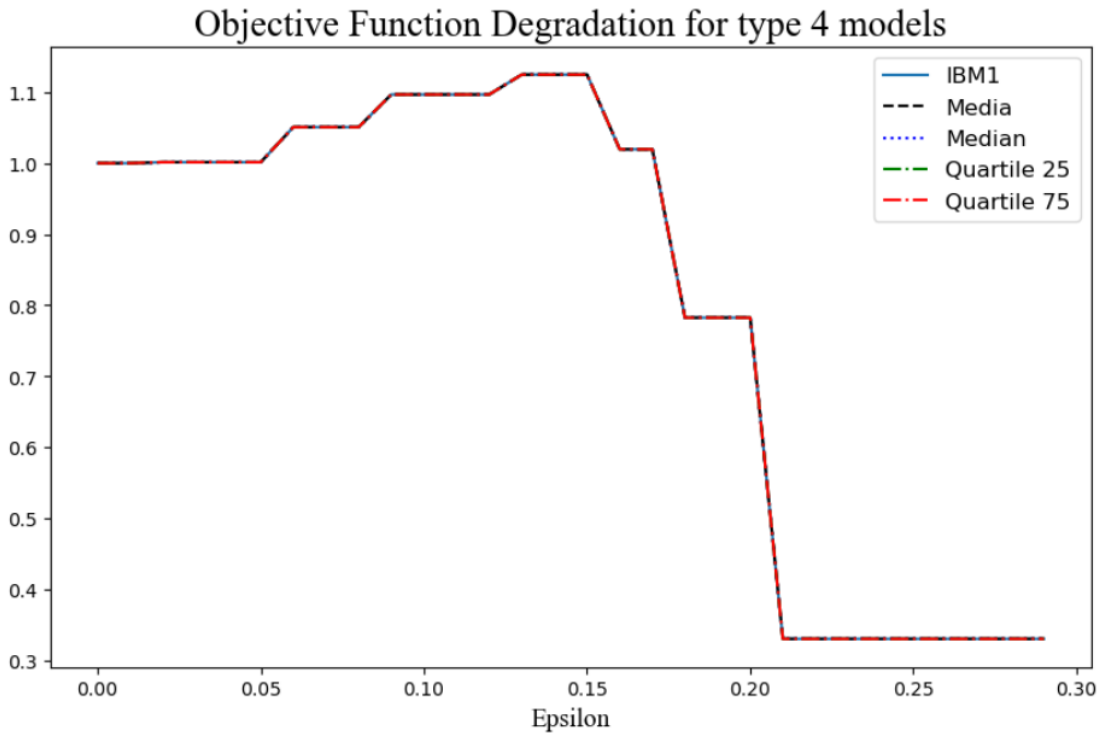


Infeasibility from type 3 problems.

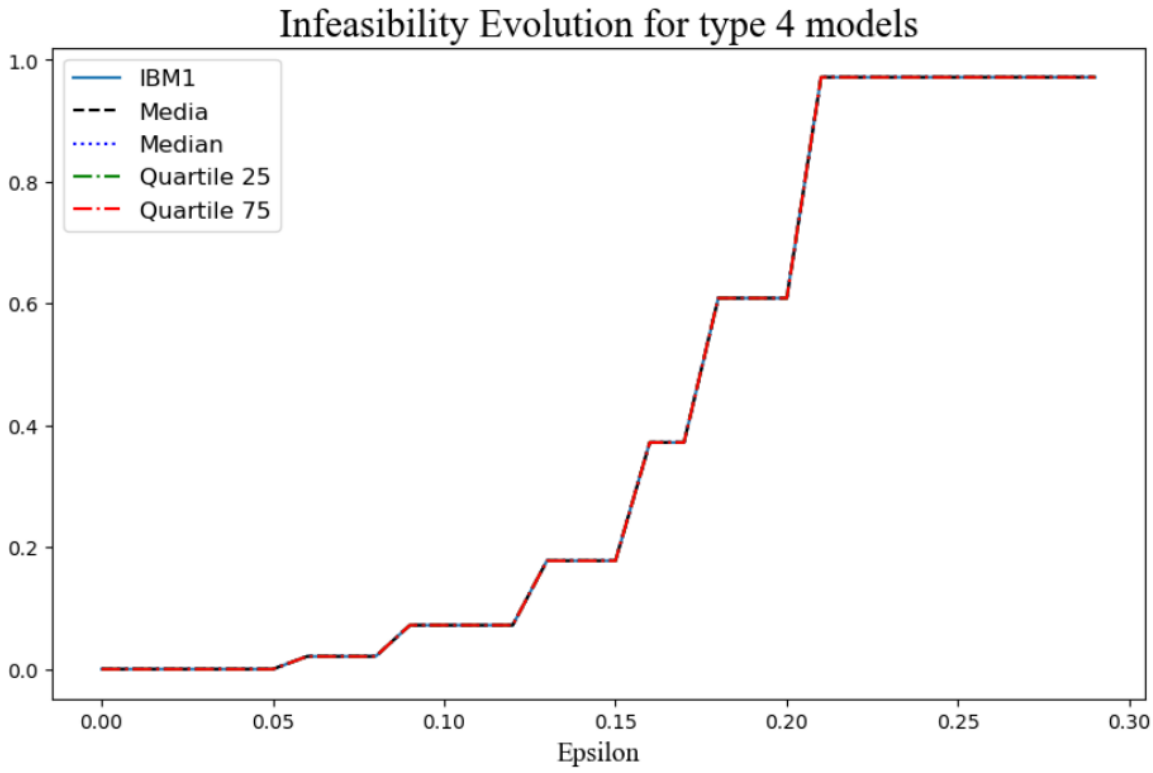


Complexity from type 3 problems.

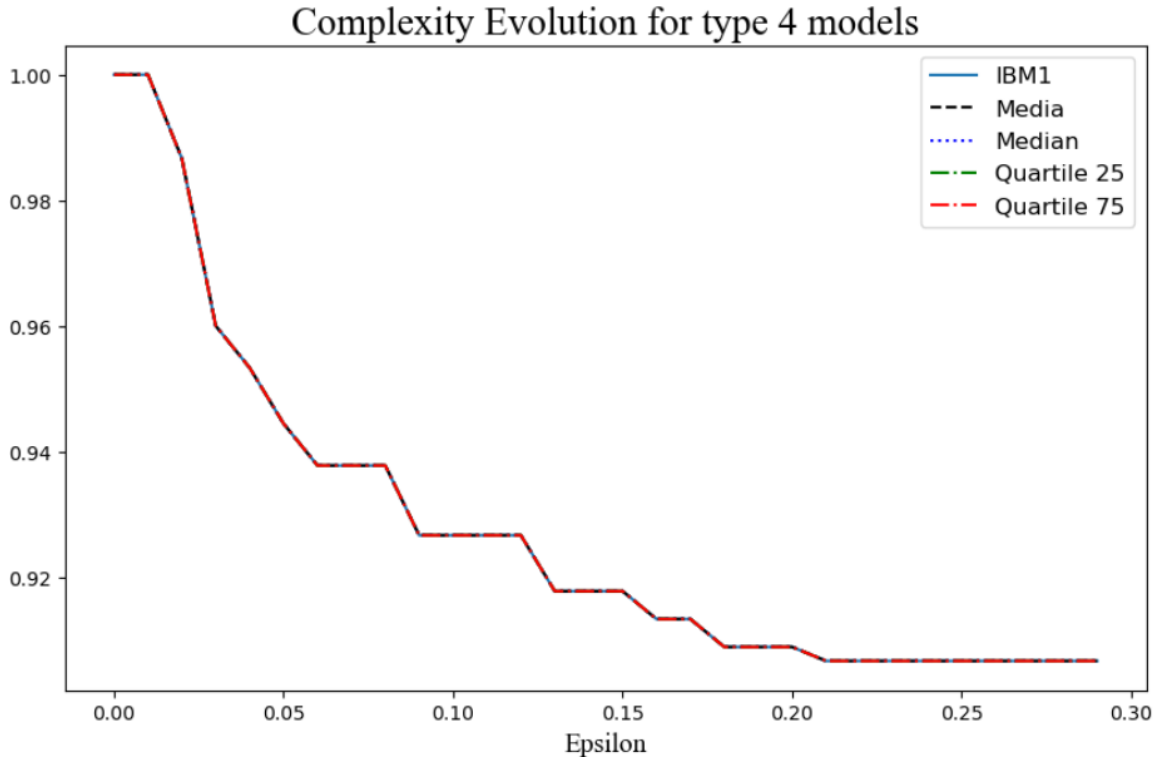
Finally, the fourth behaviour is the one that we expected before doing the analysis. For these problems, the complexity of the problem starts decreasing from the beginning, while the unfeasibility and the objective function are minimally affected. The simplification here has an important role, because the problem can be solved easier, while almost keeping the optimal solution, and without turning it into an infeasible model. Even though only 1 model follows this behaviour, it is the most important one, because it shows that it is possible to simplify problems and still get almost the optimal solution, without being infeasible. Obviously for this one, all the statistics match with the values of the only model.



Objective functions from type 4 problems.



Infeasibility from type 4 problems.



Complexity from type 4 problems.

5.3. Conclusions of the analysis

After visualizing and analysing the graphs the next conclusions are obtained:

Firstly, the sensibility analysis show that the problems react mainly in four different ways, and only in one of them makes sense doing the simplification. With this method, any problem can be analysed graphically in a way that you can see if the simplification process will help and will make easier solving the problem.

Another conclusion is that the time that takes the algorithm to solve the simplification for each value of epsilon is not a valuable data for this battery of problems, because the difference between them is insignificant. It could only be interesting for making comparisons between different problems. Even though the battery is composed of complex problems, real life problems sometimes will be more complex and bigger, so for them the time that takes to solve for the different values of epsilon might be useful.

In addition, there are many possible simplification operations, but this analysis only considers the *sparsification* method. Taking this into account, the majority of the problems become infeasible rapidly when simplifying, what means that this simplification is of no use for those problems.

In the end, the objective of the analysis was to portrait graphically how the simplifications affect the battery of problems, and with the coding made this objective is fulfilled.

6. Bibliography

Bridgelall, R. (2023). *Tutorial and Pratice in Linear Programming.*

Dantzig, G. B. (n.d.). *Linear Programming.*

Karloff, H. (n.d.). Linear Programming.

Ploskas, N., & Samaras, N. (n.d.). Springer Optimization and its applications.

7. Annex

7.1. Coding[3]

Uploading data from json:

```
# Defining the function to upload data from JSON
def cargar_datos_desde_json(ruta_archivo):
    with open(ruta_archivo, 'r') as archivo:
        datos = json.load(archivo)
    return datos
```

Reading and showing data:

```
# Reading and showing the result of an specific model
model_name = 'ORANI'  # Name of the model
tipo = 'primal' # primal or dual
key = 'epsilon' # key considered
MODELO_pr_eps = datos[model_name][tipo][key]
print(MODELO_pr_eps)
```

Objective function normalized:

```
Model_name = 'MODELO'
tipo = 'primal'
key = 'of_original_decision'
MODELO_pr_ofod = datos[model_name][tipo][key]

divider = MODELO_pr_ofod[0]
# Using a loop to divide each value by the divider
MODELO_pr_ofod_pu = []
for elemento in MODELO_pr_ofod:
    MODELO_pr_ofod_pu.append(elemento / divider)
```

[3] Open AI and Microsoft Copilot were used to optimize the code and for error solving during coding.

Infeasibility index:

```
# Reading and showing the result of an specific model
model_name = 'IBM1'  # Name of the model
tipo = 'primal' # primal or dual
key = 'epsilon' # key considered
modelo_pr_eps = datos_buenos[model_name][tipo][key]

model_name = 'IBM1'  # Name of the model
tipo = 'primal' # primal or dual
key = 'constraint_violation' # key considered
modelo_pr_cv = datos_buenos[model_name][tipo][key]

model_name = 'IBM1'  # Name of the model
tipo = 'dual' # primal or dual
key = 'decision_variables' # key considered
modelo_du_dv = datos_buenos[model_name][tipo][key]

model_name = 'IBM1'  # Name of the model
tipo = 'primal' # primal or dual
key = 'of_original_decision' # key considered
modelo_pr_ofod = datos_buenos[model_name][tipo][key]

##INFEASIBILITY INDEX
cifra_referencia = 1e-6

modelo_pr_cv_sin_nan = quitar_sublistas_nan(modelo_pr_cv)
modelo_pr_cv_filtrado =
establecer_a_cero_valores_menores(modelo_pr_cv_sin_nan,
cifra_referencia)

producto_cv_vd = multiplicar_matrices(modelo_pr_cv_filtrado,
modelo_du_dv)
suma_producto = obtener_sumas_de_sublistas(producto_cv_vd)

unfeasiblity_index = [x / abs(modelo_pr_ofod[0]) for x in
suma_producto]
```

Complexity index:

```
## PROBLEM COMPLEXITY

## Cálculo de la media de la constraint violations
modelo_pr_cv_medias = calcular_medias(modelo_pr_cv)

## Cálculo de los índices cambiados a 0 de la matriz A
acumulado_modelo_ci = calcular_longitudes(modelo_pr_ci)
elementos_A_totales = len(modelo_pr_dv) * len(modelo_du_dv)

## Ahora calculamos el número de no 0s en la matriz A, para cada valor
de epsilon. Esto lo podemos calcular mediante el
## número de índices que cambian en cada nivel de epsilon
elementos_A_que_se_hacen_0_pu = [x / elementos_A_totales for x in
acumulado_modelo_ci]
complexity_problem = [1 - x for x in elementos_A_que_se_hacen_0_pu]
```

Generalized function:

```
def analisis_de_sensibilidad(modelo, datos):
    if modelo in datos:
        modelo_datos = datos[modelo]
        modelo_primal = modelo_datos.get('primal', {})
        modelo_dual = modelo_datos.get('dual', {})

        modelo_pr_eps = modelo_primal.get('epsilon', [])
        modelo_pr_of = modelo_primal.get('objective_function', [])
        modelo_pr_dv = modelo_primal.get('decision_variables', [])
        modelo_pr_ci = modelo_primal.get('changed_indices', [])
        modelo_pr_cv = modelo_primal.get('constraint_violation', [])
        modelo_pr_ofod = modelo_primal.get('of_original_decision', [])
        modelo_pr_time = modelo_primal.get('execution_time', [])
        modelo_du_dv = modelo_dual.get('decision_variables', [])

        # Cálculo de degradación de la función objetivo original
        divisor = modelo_pr_of[0] if modelo_pr_of else None
        modelo_pr_of_pu = [elemento / divisor for elemento in
modelo_pr_of] if divisor else None
        # Cálculo de degradación de la función objetivo original
        divisor1 = modelo_pr_ofod[0] if modelo_pr_ofod else None
        modelo_pr_ofod_pu = [elemento / divisor1 for elemento in
modelo_pr_ofod] if divisor1 else None


        ## Cálculo de INFEASIBILITY INDEX
        cifra_referencia = 1e-6
        modelo_pr_cv_sin_nan = quitar_sublistas_nan(modelo_pr_cv)
        modelo_pr_cv_filtrado =
establecer_a_cero_valores_menores(modelo_pr_cv_sin_nan,cifra_referenci
a)

        producto_cv_vd =
multiplicar_matrices(modelo_pr_cv_filtrado,modelo_du_dv)
        suma_producto = obtener_sumas_de_sublistas(producto_cv_vd)
        infeasiblity_index  = [x / abs(modelo_pr_ofod[0]) for x in
suma_producto]

        ## Cálculo de PROBLEM COMPLEXITY

        ## Cálculo de la media de la constraint violations
        modelo_pr_cv_medias = calcular_medias(modelo_pr_cv)
        ## Cálculo de los índices cambiados a 0 de la matriz A
        acumulado_modelo_ci = calcular_longitudes(modelo_pr_ci)
        #for i, num in enumerate(acumulado_modelo_ci[1:], start=1):
        #    if num == 0:
        #        acumulado_modelo_ci[i] = float('nan')
        convert_late_zeros_to_nan(acumulado_modelo_ci)
        elementos_A_totales = len(modelo_pr_dv)*len(modelo_du_dv)
        ## Ahora calculamos el número de no 0s en la matriz A, para
cada valor de epsilon. Esto lo podemos calcular mediante el
        ## número de indices que cambian en cada nivel de epsilon
        elementos_A_que_se_hacen_0_pu = [x / elementos_A_totales for x
in acumulado_modelo_ci]
        complexity_problem = [1 - x for x in
elementos_A_que_se_hacen_0_pu]
        suma_objfunc_unfeasiblity = [a + b for a, b in
zip(modelo_pr_ofod_pu, infeasiblity_index)]

        ## PROBLEM COMPLEXITY, con el tiempo de ejecución
        #modelo_pr_time1 = modelo_pr_time[1:];
```

```
        # Convertir los tiempos a números flotantes
        #modelo_pr_time1 = [float(tiempo.split(':')[2]) for tiempo in
modelo_pr_time1]

        #divisor = modelo_pr_time1[0] if modelo_pr_time1 else None
        #modelo_pr_time_pu = [elemento / divisor for elemento in
modelo_pr_time1] if divisor else None
        #complexity_problem = modelo_pr_time_pu;
        ## GRÁFICAS
        #titulo1=(modelo + " Objective function degradation vs
complexity")
        #titulo2=(modelo + " Infeasibility ")
        #titulo3=(modelo + " Sum of infeasibility and objective
function")
        titulo1=(modelo + " Objective function degradation")
        titulo2=(modelo + " Infeasibility evolution")
        titulo3=(modelo + " Complexity evolution")
        objective_function = "Objective function"
        complexity = "Complexity"
        infeasibility = "Infeasibility"
        #suma = "Objective function + Infeasibility"

#graficar2(modelo_pr_eps,modelo_pr_ofod_pu,complexity_problem,titulo1,
objective_function,complexity)
        #graficar2(modelo_pr_eps,infeasiblity_index,
complexity_problem,titulo2,infeasibility,complexity)

graficar1(modelo_pr_eps,modelo_pr_ofod_pu,titulo1,objective_function)

graficar1(modelo_pr_eps,infeasiblity_index,titulo2,infeasibility)
        graficar1(modelo_pr_eps,complexity_problem,titulo3,complexity)

        print(elementos_A_que_se_hacen_0_pu)
        return

    else:
        print(f"El modelo '{modelo}' no se encontró en los datos
proporcionados.")
        return None
```

Loop for iterating the battery:

```
for elemento in modelos:

    analisis_de_sensibilidad(elemento, datos_buenos)
```

Function used to graph the different type of problems together:

```
def calcular_media(lista):
    return np.nanmean(lista) if lista else None

def calcular_mediana(lista):
    return np.nanmedian(lista) if lista else None

def calcular_cuartiles(lista):
    return np.nanpercentile(lista, [25, 75]) if lista else (None,
None)
```

```
def ajustar_longitudes(eps, datos):
    min_longitud = min(len(eps), len(datos))
    return eps[:min_longitud], datos[:min_longitud]

def rellenar_con_nan(datos, longitud_maxima):
    return datos + [np.nan] * (longitud_maxima - len(datos))

def analisis_de_sensibilidad_global(modelos, datos):
    # Definir variables para almacenar los datos
    all_epsilons = {}
    all_objective_function_degradation = {}
    all_infeasibility = {}
    all_complexity = {}

    # Inicializar listas para acumular datos
    objective_function_degradation_list = []
    infeasibility_list = []
    complexity_list = []

    # Determinar la longitud máxima
    max_length = 0
    for modelo in modelos:
        if modelo in datos:
            modelo_datos = datos[modelo]
            modelo_primal = modelo_datos.get('primal', {})

            modelo_pr_eps = modelo_primal.get('epsilon', [])
            modelo_pr_of = modelo_primal.get('objective_function', [])
            modelo_pr_cv = modelo_primal.get('constraint_violation',
[])
            modelo_pr_ci = modelo_primal.get('changed_indices', [])
            modelo_pr_ofod = modelo_primal.get('of_original_decision',
[])
            modelo_du_dv = modelo_datos.get('dual',
{}).get('decision_variables', [])

            # Cálculo de degradación de la función objetivo original
            divisor = modelo_pr_of[0] if modelo_pr_of else None
            modelo_pr_ofod_pu = [elemento / divisor for elemento in
modelo_pr_of] if divisor else []

            # Cálculo de INFEASIBILITY INDEX
            cifra_referencia = 1e-6
            modelo_pr_cv_sin_nan = quitar_sublistas_nan(modelo_pr_cv)
            modelo_pr_cv_filtrado =
establecer_a_cero_valores_menores(modelo_pr_cv_sin_nan,
cifra_referencia)

            if modelo_du_dv and modelo_pr_cv_filtrado:
                producto_cv_vd =
multiplicar_matrices(modelo_pr_cv_filtrado, modelo_du_dv)
                suma_producto =
obtener_sumas_de_sublistas(producto_cv_vd)
                infeasiblity_index = [x / abs(modelo_pr_ofod[0]) for x
in suma_producto] if modelo_pr_ofod else []
            else:
                infeasiblity_index = []

            # Cálculo de PROBLEM COMPLEXITY
            acumulado_modelo_ci = calcular_longitudes(modelo_pr_ci)
            convert_late_zeros_to_nan(acumulado_modelo_ci)
```

```
            elementos_A_totales = len(modelo_pr_dv) *
len(modelo_du_dv)
            elementos_A_que_se_hacen_0_pu = [x / elementos_A_totales
for x in acumulado_modelo_ci] if elementos_A_totales else []
            complexity_problem = [1 - x for x in
elementos_A_que_se_hacen_0_pu]

            # Actualizar la longitud máxima
            max_length = max(max_length, len(modelo_pr_eps),
len(modelo_pr_ofod_pu), len(infeasiblity_index),
len(complexity_problem))

            # Acumular datos para análisis global
            all_epsilons[modelo] = modelo_pr_eps
            all_objective_function_degradation[modelo] =
modelo_pr_ofod_pu
            all_infeasibility[modelo] = infeasiblity_index
            all_complexity[modelo] = complexity_problem

            # Asegurarse de que todas las listas tengan la misma
longitud

objective_function_degradation_list.append(rellenar_con_nan(modelo_pr_
ofod_pu, max_length))

infeasibility_list.append(rellenar_con_nan(infeasiblity_index,
max_length))

complexity_list.append(rellenar_con_nan(complexity_problem,
max_length))
        else:
            print(f"El modelo '{modelo}' no se encontró en los datos
proporcionados.")

    # Función para calcular estadísticas por índice
    def calcular_estadisticas_por_indice(datos_por_indice):
        num_epsilons = len(datos_por_indice[0]) if datos_por_indice
else 0
        media = [calcular_media([datos[i] for datos in
datos_por_indice]) for i in range(num_epsilons)]
        mediana = [calcular_mediana([datos[i] for datos in
datos_por_indice]) for i in range(num_epsilons)]
        cuartiles = [calcular_cuartiles([datos[i] for datos in
datos_por_indice]) for i in range(num_epsilons)]
        return media, mediana, cuartiles

    # Calcular estadísticas para cada métrica
    media_of, mediana_of, cuartiles_of =
calcular_estadisticas_por_indice(objective_function_degradation_list)
    media_infeasibility, mediana_infeasibility,
cuartiles_infeasibility =
calcular_estadisticas_por_indice(infeasibility_list)
    media_complexity, mediana_complexity, cuartiles_complexity =
calcular_estadisticas_por_indice(complexity_list)

    # Función para graficar los datos
    def graficar_datos(titulo, ylabel, datos_dict,
estadisticas_dict=None):
        plt.figure(figsize=(10, 6))
        colores = plt.cm.get_cmap('tab10', len(datos_dict))
```

```
        for i, modelo in enumerate(modelos):
            eps = all_epsilons.get(modelo, [])
            datos = datos_dict.get(modelo, [])
            eps_ajustado, datos_ajustado = ajustar_longitudes(eps,
datos)

            plt.plot(eps_ajustado, datos_ajustado, label=f'{modelo}')

        if estadisticas_dict:
            num_epsilons = len(estadisticas_dict.get('media', []))
            eps_comunes = np.linspace(min(eps_ajustado),
max(eps_ajustado), num=num_epsilons)
            media = estadisticas_dict.get('media', [])
            mediana = estadisticas_dict.get('mediana', [])
            cuartiles = estadisticas_dict.get('cuartiles', [])
            cuartiles_25 = [q[0] for q in cuartiles]
            cuartiles_75 = [q[1] for q in cuartiles]

            print(f"Longitudes para graficar estadísticas:
eps_ajustado = {len(eps_ajustado)}, media = {len(media)}, mediana =
{len(mediana)}, cuartiles_25 = {len(cuartiles_25)}, cuartiles_75 =
{len(cuartiles_75)}")

            plt.plot(eps_ajustado, media[:len(eps_ajustado)], '--',
label='Media', color='black')
            plt.plot(eps_ajustado, mediana[:len(eps_ajustado)], ':',
label='Median', color='blue')
            plt.plot(eps_ajustado, cuartiles_25[:len(eps_ajustado)],
'-.', label='Quartile 25', color='green')
            plt.plot(eps_ajustado, cuartiles_75[:len(eps_ajustado)],
'-.', label='Quartile 75', color='red')

        plt.title(titulo, fontname='Times New Roman', fontsize=20)
        plt.xlabel("Epsilon", fontname='Times New Roman', fontsize=14)
        plt.ylabel(ylabel)
        plt.legend(fontsize=12)
        plt.show()

    # Graficar datos acumulados
    graficar_datos("Objective Function Degradation for type 4 models",
"", all_objective_function_degradation,
                   {'media': media_of, 'mediana': mediana_of,
'cuartiles': cuartiles_of})
    graficar_datos("Infeasibility Evolution for type 4 models", "",
all_infeasibility,
                   {'media': media_infeasibility, 'mediana':
mediana_infeasibility, 'cuartiles': cuartiles_infeasibility})
    graficar_datos("Complexity Evolution for type 4 models", "",
all_complexity,
                   {'media': media_complexity, 'mediana':
mediana_complexity, 'cuartiles': cuartiles_complexity})

    return
```

Models distributed in different types:

```
modelos_tipo1 = ['AIRSP', 'AMPL','ASYNCLOOP','PAPERCO','CHINA',
'CLEARLAK', 'DEMO1', 'JOBT', 'LANDS', 'MARKOV', 'MEXSS', 'MINE',
'PAKLIVE', 'PRODMIX', 'ROBERT', 'SPARTA', 'SRPCHASE']
#graphed:
```

```
modelos_tipo1 = ['AIRSP', 'PRODMIX', 'SPARTA' ]
modelos_tipo2 = ['AIRCRAFT', 'BLEND','DIET', 'KAND','SRKANDW','UIMP']
#graphed:
modelos_tipo2 = ['AIRCRAFT','SRKANDW','UIMP']
modelos_tipo3 = ['DEA','GUSSEX1','GUSSGRID','SENSTRAN','TRNSPORT']
#grahped:
modelos_tipo3 = ['GUSSEX1','GUSSGRID','SENSTRAN','TRNSPORT']
modelos_tipo4 = ['IBM1']
```

Auxiliary functions used:

```
#Function to calculate the media of a list.
def calcular_medias(lista):
    medias = []
    for sublista in lista:
        if isinstance(sublista, list):
            if len(sublista) > 1:
                media = np.nanmean(sublista)
                medias.append(media)
            elif len(sublista) == 1:
                medias.append(sublista[0])
            else:
                medias.append(None)
        elif isinstance(sublista, (float, int)):
            medias.append(sublista)
    return medias
```

```
def calcular_longitudes(vector_con_vectores):
    # Creamos una lista para almacenar las longitudes de los vectores
    longitudes = []

    # Iteramos sobre cada vector en el vector_con_vectores
    for vector in vector_con_vectores:
        # Si el vector es None, consideramos su longitud como 0
        if vector is None:
            longitudes.append(0)
        # Si el vector contiene NaN, lo excluimos del cálculo de
longitud
        elif isinstance(vector, (list, np.ndarray)) and
np.isnan(vector).any():
            continue
        else:
            # Verificamos si el vector es iterable antes de intentar
calcular su longitud
            try:
                longitud = len(vector)
            except TypeError:
                # Si no es iterable, agregamos 0 a las longitudes
                longitud = 0
            longitudes.append(longitud)

    return longitudes
```

```
def establecer_a_cero_valores_menores(lista_de_listas,
cifra_referencia):

    for i in range(len(lista_de_listas)):
        for j in range(len(lista_de_listas[i])):
            if abs(lista_de_listas[i][j]) < cifra_referencia:
                lista_de_listas[i][j] = 0
```

```
    return lista_de_listas


# PARA ALGUNOS MODELOS COMO AIRCRAFT, APARECEN SUBLISTAN NaN y hay que
quitarlas
def quitar_sublistas_nan(lista_de_listas):
    lista_sin_nan = [sublista for sublista in lista_de_listas if not
np.any(np.isnan(sublista))]
    return lista_sin_nan
```

```
def multiplicar_matrices(A, B):
    # Obtener las dimensiones de las matrices
    filas_A = len(A)
    columnas_A = len(A[0])
    filas_B = len(B)
    columnas_B = len(B[0])

    # Verificar si las matrices son multiplicables
    if columnas_A != columnas_B:
        raise ValueError("Las matrices no son multiplicables")

    # Inicializar la matriz resultante con ceros
    C = [[0] * columnas_B for _ in range(filas_A)]

    # Multiplicar elemento por elemento y sumar los resultados
    for i in range(filas_A):
        for j in range(columnas_B):
            for k in range(columnas_A):
                C[i][j] = A[i][j] * B[i][j]


    return C
def obtener_sumas_de_sublistas(lista_de_listas):
    return [sum(map(float, sublista)) for sublista in lista_de_listas]
```

```
def graficar1(modelo_pr_epsilon, vector1, titulo, nombre1):
    # Determinar la longitud máxima de los vectores
    max_length = max(len(modelo_pr_epsilon), len(vector1))

    # Generar el eje_x con la misma longitud que el vector más largo
    eje_x = modelo_pr_epsilon[:max_length]

    # Graficar los vectores
    plt.figure(figsize=(8, 6))
    plt.plot(eje_x[:len(vector1)], vector1, label=nombre1)

    # Configurar etiquetas y título con tamaños de letra
    plt.xlabel('Epsilon', fontsize=15,fontname = 'Times New Roman')
    plt.ylabel('', fontsize=12)
    plt.title(titulo, fontsize=16, fontname = 'Times New Roman')

    # Configurar leyenda con tamaño de letra
    plt.legend(fontsize=12)

    # Configurar tamaño de letra de los ticks
    plt.tick_params(axis='both', which='major', labelsize=12)

    # Configurar otros parámetros del gráfico
```

```
    plt.grid(False)
    plt.xlim(eje_x[0], eje_x[-1])  # Ajustar límites del eje x

    # Mostrar el gráfico
    plt.show()
```