Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería de Organización Industrial

Linear Programming with SoPlex, SoPlex complexity and SCIP complexity

Autor:

Luis Pérez, Aurora

Responsable de Intercambio en la Uva:

Mª Carmen Quintano Pastor

Universidad de destino:

Hochschule Magdeburg-Stendal

Valladolid, agosto de 2016.

TFG REALIZADO EN PROGRAMA DE INTERCAMBIO

TÍTULO: **Linear Programming with Soplex, Soplex complexity and SCIP complexity**

ALUMNO: **Aurora Luis Pérez**

FECHA: 21/07/2016

CENTRO: **Hochschule Magdeburg-Stendal (Elektrotechnik Institut)**

TUTOR: **Albert Seidl**

Resumen

El objetivo de esta investigación es conocer el modo de operación de SoPlex, un programa de resolución de problemas de programación lineal, y la evaluación de su rendimiento. También se hace un estudio muy básico de SCIP, que resuelve problemas de programación entera mixta.

Primero se expone una visión general de la investigación de operaciones y luego la investigación se centra en la programación lineal. Como se quiere hacer un estudio del funcionamiento de SoPlex y SCIP, posteriormente también se trata la complejidad computacional.

Una vez expuesta la base teórica de la investigación, primero se explica la instalación de los programas (SCIP Optimization Suite). En el estudio hecho sobre Soplex, se explican los diferentes formatos que pueden ser utilizados y para analizar su rendimiento se utilizan problemas típicos de programación lineal. Para el estudio de SCIP se emplea un ejemplo incluido en la instalación, el problema de las n Reinas.

Palabras clave

SCIP, SoPlex, Programación Lineal (PL), optimización, tiempo de ejecución.

Abstract

The purpose of this research is to know how is the operation of SoPlex, a linear programming solver develop by The Zuse Institute Berlin (ZIB), and the evaluation of the solver performance. Furthermore, it is done a basic study of the mixed integer programming solver develop by ZIB, SCIP.

First it is exposed a general view of the operation research and later it is focus on linear programming. As it is wanted to do a performance study of SoPlex and SCIP, later it is exposed in which parameters are based the computational complexity and the different class of problems depending on complexity.

Once it is presented the theoretical basic of this research, it is explained how install SCIP Optimization Suite (where both solvers are included). In Soplex study, it is explained the different formats that the solver could solve. To do the complexity study, it is used some typical linear programming problems. The study of SCIP complexity is done under an example included in the installation of the program (the Queens problem).

Keywords

SCIP, SoPlex, Linear Programming (LP), optimization, execution time.

Universidad de Valladolid

Bachelor Thesis

Business Engineering

Linear Programming with SoPlex, SoPlex complexity and SCIP complexity

Author:

Luis Pérez, Aurora

Tutor:

Seidl, Albert
Electrical Engineering Department

Magdeburg, July 2016.

Index

1. Introduction

Every time Operations Research is more used in a fields variety of industry and business. Thus, it is needed the use of methods and techniques that can give optimal or acceptable solution to the problems that arise, using different kind of techniques and solvers.

The purpose of this research is to know how is the operation of SoPlex, a linear programming solver develop by The Zuse Institute Berlin (ZIB), and the evaluation of the solver performance. Furthermore, it is done a basic study of the mixed integer programming solver develop by ZIB, SCIP.

For that reason, first it is exposed a general view of the operation research, with the different tools and techniques, and later it is focus on linear programming. It is exposed about the mathematical model language and some examples of these kind of problems.

As it is wanted to do a performance study of SoPlex and SCIP, later it is exposed in which parameters are based the computational complexity and the different class of problems depending on complexity.

Once it is presented the theoretical basic of this research, it is explained how install SCIP Optimization Suite (where both solvers are included) and the steps to start running the program.

In Soplex study, it is explained the different formats that the solver could solve and the way to do that. To do the complexity study, it is used some typical linear programming problems.

At the end, the study of SCIP complexity is done under an example included in the installation of the program.

2. Operation Research

Since the advent of the industrial revolution, the world has seen a remarkable growth in the size and complexity of organizations. It has been an increase in the division of labour and segmentation of management responsibilities in these organizations. However, it has created new problems as to allocate the available resources to the various activities in a way that is most effective for the organization as a whole. These problems and the need to find a better way to solve them provided the environment for the emergence of Operations Research [1][4].

Operation Research could be defined as modern discipline that uses mathematical, statistical and algorithms for modelling and solving complex problems, determining the optimal solution and improving decision-making [2].

The Operation Research started just before World War II in Britain with the establishment of teams of scientists to study the strategic and tactical problems involved in military operations. The objective was to find the most effective utilization of limited military resources by the use of quantitative techniques [1][4].

When the war ended, the success of Operation Research in the war effort spurred in applying it outside the military as well. By the early 1950s, it had been introduced the use of Operation Research to a variety of organizations in business, industry, and government. Two of the factors that played a key role in the rapid growth of Operation Research were the substantial progress in improving techniques and the computer revolution, with their ability to perform arithmetic calculations millions of times faster than a human being can [1][4].

2.1. Operation Research Process

The stages of development of Operation Research are:

- Observe the problem environment.
- Analyse and define the problem.
- Develop a model.
- Select appropriate data input.
- Provide a solution and test its reasonableness.
- Implement the solution.

The first step in the process of Operation Research is the problem environment observation. Included different activities (conferences, site visit, research, observations…) will provide sufficient information to formulate the problem.

Once the problem has been observed, it will be analysed and defined the problem. Furthermore, it must be defined the objectives, uses and limitations of the problem. The outputs of this step are clear grasp of need for a solution and its nature understanding.

After the model, a representation of some abstract or real situation, must be developed. The models are mathematical models, which describe systems, processes in the form of equations and relationships. The model is tested in the field under different environmental constraints and modified in order to work. Sometimes the model is modified to satisfy the management with the results.

The model will work appropriately when there is appropriate data input. For that reason, it must be necessary analysed internal and external data, fact analysis and opinions, using computer data banks.

With the help of the model and the input data, it will be got a solution. Instead of implemented the solution, it is used to test the model and to find if there is any limitations. If the solution is not reasonable or the behaviour of the model is not proper, the model will be updated and modified. After this process, it will be got the solution that supports the organizational objectives and it should be implemented.

2.2. Tools and Techniques

To solve the Operation Research models there is a lot of tools and techniques available. The most important techniques are exposed below.

1) Linear Programming

Linear Programming is a method that try to find the best possible solution in allocating limited resources to achieve maximum profit or minimum cost. However, it is applicable only where all relationships are linear.

There are different methods available to solve linear programming problems. The most used are pivotal algorithms, particularly simplex algorithm.

This method is exposed in the section 3 with more details.

2) Network scheduling

This technique is a special case of the more general linear program that is used to plan, schedule and monitor large projects. The aim of this technique is minimize trouble spots (such as delays, interruptions, production bottlenecks, etc.) by identifying the critical factors.

The different activities and their relationships of the project are represented diagrammatically with the help of networks and arrows, which is used for identifying critical activities and path.

The main types of technique in network scheduling are:

- PERT (Program Evaluation and Review Technique). It is used when the activities time is not known accurately; only probabilistic estimate of time is available.
- CPM (Critical Path Method). It is used when activities time is know accurately.

3) Integer Programming

An integer programming problem is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be intefgers. Most problems of practical size are very difficult or impossible to solve (NP-complexity problems exposed in the section 4).

In many settings the term refers to integer linear programming (IPL), in which the objective function and the term refers to integer linear programming (ILP), in which the objective function and the constraints are linear.

4) Non-linear Programming

The non-linear Programming is used when the objective function and the constraints are not linear. These problems, in general, are much more difficult to solve than the linear programming problems.

Most (if not all) real world applications require a non-linear model. In order to be make the problems tractable, they are often approximate using linear functions but limited to some range, because approximation becomes poorer as the range is extended.

5) Dynamic Programming

Dynamic programming is a method of analysing multistage decision processes. Each elementary decision depends on those preceding decisions and as well as external factors.

The process begins in some initial state where a decision is made. The decision causes a transition to a new state. Based on the starting state, ending state and decision return is realized. The process continues through a sequence of states until finally a final state is reached.

The problem is to find the sequence that maximizes the total return. Objectives with very general functional forms may be handled and a global optimal solution is always obtained. The number of states grows exponentially with the number of dimensions of the problem.

6) Stochastic Process

A stochastic process is simply a probability process, that is any process in nature whose evolution could be analysed successfully in terms of probability.

The model is described in part by enumerating the states in which the system can be found. The state is like a snapshot of the system at a point time that describes the attributes of the system. Events occur that change the state of the system.

7) Markov Process

Markov process is a stochastic process that permits to predict changes over time information about the behaviour of a system is known.

This is used in decision making in situations where the various states are defined. The probability form one state to another state is known and depends on the current state and is independent of how it has been arrived at that particular state.

8) Simulation

Simulation is a procedure that studies a problem by creating a model of the process involved in the problem and then through a series of organized trials and error solutions attempt to determine the best solution. Sometimes this a difficult consuming procedure.

Simulation is used when actual experimentation is not feasible or solution of model is not possible.

9) Inventory Theory

Inventories are materials stored, waiting for processing or experiencing processing. Inventory model make a decisions that minimize total inventory cost. This model successfully reduces the total cost of purchasing, carrying, and out of stock inventory.

10) Game Theory

The game theory is a set of concepts aimed at decision making in situations of competition and conflict (as well as of cooperation and interdependence) under specified rules.

It employs games of strategy but not of chance. A strategic game represents a situation where two or more participants are faced with choices of action, by which each may gain or lose, depending on what others choose to do or not to do. The final outcome of a game, therefore, is determined jointly by the strategies chosen by all participants.

11) Decision Theory

Decision theory is concerned with making decisions under conditions of complete certainty about the future outcomes and under conditions such that we can make some probability about what will happen in future.

Decision theory can be broken into two branches: normative and descriptive. Normative decision theory gives advice on how to make the best decisions, given a set of uncertain beliefs and a set of values. Descriptive decision theory analyses how existing, possibly irrational, agents actually make decisions.

It is closely related to the field of game theory. Decision theory is concerned with the choices of individual agents whereas game theory is concerned with interactions of agents whose decisions affect each other.

12) Queuing Theory

Queuing theory is the mathematical study of waiting lines or queues. A model is constructed so that queue lengths and waiting time can be predicted. The objective is minimizing the cost of waiting without increasing the cost of servicing.

13)Heuristics

Heuristics are techniques that are looking for a good solution with high quality at a reasonable computational cost, though without guarantee an optimal solution. In general, not even the degree of error are known.

Heuristics could be classified according to the methods used:

- Construction methods. It is generated a solution (from an empty solution) by adding components to complete the solution.
- Decomposition methods. The problem is dividing into smaller and the solution is obtained from the solution of each small problem.
- Reduction methods. It is tried to identify some characteristic of the solution to simplify the treatment of the problem.
- Inductive methods. It is gotten a solution to the original problem from another simplified problem.
- Local search methods. It is started from a solution and iteratively will replace the current solution with a similar solution of better qualiy.

14)Metaheuristics

A metaheuristic is a higher-level heuristic designed to find, generate, or select a heuristic that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computational capacity.

The most common techniques are:

- Genetic algorithms. They are adaptive methods based on the genetic process of living organism. They are mathematical functions or software routines that takes as inputs the outputs copies and returns as which of them should generate offspring for the next generation.
- Simulated annealing. It is based on the physic metal heating. The idea is allowing movements to solutions that deteriorate the objective function to escape to local optima.

- Tabu search. It uses a local search with short-term memory that allows escaping from local minima and avoiding cycles.
- Ant colony optimization. It is based on the ants behaviour. While each individual ant has basic capabilities, the colony achieved together intelligent behaviour. Despite being almost blind insects manage to find the shortest path between the nest and a food source and return, based on a pheromones communications (leaving a “trail” that serves as a reference to others.

2.3. Applications of Operation Research

There are voluminous of applications of Operation Research. The following are a sum up of typical operations research applications to show how widely there techniques are used today in different areas [3]:

Area	Applications
Accounting	- Assigning audit teams effectively - Credit policy analysis - Cash flow planning - Developing standard costs - Planning of delinquent account strategy
Construction	- Project scheduling, monitoring and control - Determination of proper work force - Deployment of work force - Allocation of resources to projects
Facilities Planning	- Factory location and size decision - Estimation of number of facilities required - Hospital planning - International logistic system design - Transportation loading and unloading - Warehouse location decision
Finance	- Building cash management models - Allocating capital among various alternatives - Building financial planning models - Investment analysis - Portfolio analysis - Dividend policy making
Manufacturing	- Inventory control - Marketing balance projection - Production scheduling - Production smoothing

Marketing	- Advertising budget allocation - Product introduction timing - Selection of Product mix - Deciding most effective packaging alternative
Human Resources	- Personnel planning - Recruitment of employees - Skill balancing - Training program scheduling - Designing organizational structure more effectively
Purchasing	- Optimal buying - Optimal reordering - Materials transfer
Research and developing	- R & D Projects control - R & D Budget allocation - Planning of Product introduction

2.4. Limitations of Operation Research

Operations Research has certain limitations, that are mostly related to the model building and money and time factors problems involved in its application. Some of them are as given below:

- Often, it is necessary to simplify the original problem to manipulate and obtain a solution.
- Most models only consider a single target and often, in organizations, have multiple objectives.
- There is a tendency not to consider all restrictions on a practical problem, because the methods of teaching and training give the application of this science centrally based on small problems for reasons of practical nature.
- Almost never cost-benefit analysis of the solution implementation is done by the Operation Research. Sometimes the potential benefits are outweighed by the costs incurring in the development and implementation of a model.

3. Linear Programming

A linear programming problem may be defined as the problem of maximizing or minimizing a linear function subject to linear constraints. The constraints may be equalities, inequalities or both [5].

The main elements of any constrained optimization problem are variables, the objective function, constraints and variable bounds.

The variables, also called decision variables, usually represent things that you can adjust or control and its value are not known when the problem is started. The variables could be represented like $x_1, x_2, \dots, x_n$. The goal is to find values of the variables that provide the best value of the objective function.

The objective function is a mathematical expression that combines the variables to express your goal. You will be required to either maximize or minimize the objective function. It could be written as:

$$z = c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n \cdot x_n$$

Where $c_1, c_2, \dots, c_n$ are known real numbers called cost coefficients.

The constraints are mathematical expressions that combine the variables to express limits on the possible solutions. The way to express the constraints are:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \quad &\leq, =, \geq \quad b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \quad &\leq, =, \geq \quad b_2 \\ \dots \qquad\qquad\qquad\qquad\qquad &\dots \qquad \dots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \quad &\leq, =, \geq \quad b_m \end{aligned}$$

Where every restriction could be $\leq, =, \geq$. The a_{ij} coefficient is called the technological coefficient and b_i is the independent term.

Only rarely are the variables in an optimization problem permitted to take on any value from minus infinity to plus infinity. Instead, the variables usually have bounds. If the bounds are only $x_1 \geq 0$, $x_2 \geq 0, \dots x_n \geq 0$, it is called non-negativity constraints, but sometimes the variables have an upper bound (u_j), a lower bound (l_j), or both:

$$u_j \geq x_j \geq l_j$$

To end, the R^n point that satisfied all the constraints is called space restrictions or feasible set.

The Standard Maximun Problem should find a n-vector, $x = (x_1, \dots, x_n)$ to maximize the objective function:

$$maximize\ z = c_1 \cdot x_1 + \cdots + c_n \cdot x_n$$

subject to the constraints

$$\begin{array}{l} a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1 \\ a_{21}x_1 + \cdots + a_{2n}x_n \leq b_2 \\ \ldots \qquad\qquad \ldots \qquad \ldots \\ a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m \end{array} \qquad (or\ Ax \leq b)$$

and

$$x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0$$

The Standard Minimum Problem tries to find a n-vector, but in that case, to minimize the objective function:

$$minimize\ z = c_1 \cdot x_1 + \cdots + c_n \cdot x_n$$

subject to the constraints

$$\begin{array}{l} a_{11}x_1 + \cdots + a_{1n}x_n \geq b_1 \\ a_{21}x_1 + \cdots + a_{2n}x_n \geq b_2 \\ \ldots \qquad\qquad \ldots \qquad \ldots \\ a_{m1}x_1 + \cdots + a_{mn}x_n \geq b_m \end{array} \qquad (or\ Ax \geq b)$$

and

$$x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0$$

Note that the main constraints are written as ≤ for the standard maximum problem and ≥ for the standard minimum problem.

Linear programming problems could be solved with a graphic if there is two or three variables. If there is more than three variables, the common way to solve the problems is the Simplex Method (explained in 3.2. section).

3.1. Linear Programming Problems

The linear programming could be used with different problems to find an optimal solution. Then it is formulated the mathematical linear programming model of different problems.

3.1.1. Product-Mix Problem

One of the classic applications of Linear Programming models is the product mix problem. We consider n products involved in the production process and m resources

which has limited quantities. The goal is to find the production schedule that maximizes benefits while the amounts of resources avaible are not exceeded and demand is satisfied.

The parameters involved are:

$n =$ Number of products in the production process.

$m =$ Number of resources that are available.

$p_j =$ Selling prize of product *j,* with $j = 1, \dots, n$

$c_j =$ Cost of a unit of product *j,* with $j = 1, \dots, n$

$b_i =$ Quantity of resource i available during the period, with $i = 1, \dots, m$

$a_{ij} =$ Units or quantity of resource i required to produce a unit of product j.

$u_j =$ Maximum potential sales of product j in the period considered.

$l_j =$ Minimum potential sales of product j in the period considered ($l_j \geq 0 \ \ \forall j = 1, \dots, n$).

The decision variables will be:

$x_j =$ Quantity to produce of product j in the period considered.

The mathematical model to solve will be:

$$maximize \quad \sum_{j=1}^{n} (p_j - c_j) \cdot x_j$$

$$Subject\ to \quad \sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i \qquad \forall i = 1, \dots, m$$

$$x_j \leq u_j \qquad \forall j = 1, \dots, n$$

$$x_j \geq l_j \qquad \forall j = 1, \dots, n$$

The objective function try to maximize the benefit, that it is calculated as the sum of the profit margin of each product (prize minus cost) by the product.

The first constraint is the capacity restriction and the others are the bounds of the variables.

3.1.2. Transport Problem

The transportation problem is concerned with finding the minimum cost of transporting a single commodity from a m number of sources to a n number of destinations.

The data of the model include:

- The level of supply at each source and the amount of demand at each destination.
- The unit transportation cost of the commodity from each source to each destination.

Since there is only one commodity, a destination can receive its demand from more than one source. The objective is to determine how much should be shipped from each source to each destination to minimise the total transportation cost.

The parameters involved are:

$m =$ Number of sources.

$n =$ Number of destinations.

$a_i =$ Amount of supply available at source i

$b_i =$ Demand required at destination j

$c_{ij} =$ Cost of transporting one unit between source i and destination j

The decision variable is:

$x_{ij} =$ Quantity transported from source i to destination j

And the problem to solve will be:

$$minimize \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} \cdot x_{ij}$$

$$Subject\ to \quad \sum_{j=1}^{n} x_{ij} \leq a_i \quad \forall i = 1, \dots, m$$

$$\sum_{i=1}^{m} x_{ij} \geq b_i \quad \forall j = 1, \dots, n$$

$$x_{ij} \geq 0 \quad \forall i \, \forall j$$

The first constraint says that the sum of all shipments from a source cannot exceed the available supply. The second constraint specified that the sum of all shipments to a destination must be at least as large as the demand.

In the transportation problem is defined the total supply as $S_T = \sum_{i=1}^{m} a_i$, and the total demand as $D_T = \sum_{j=1}^{n} b_i$. When the total supply is equal to the total demand ($S_T = D_T$) then the transportation model is said to be balanced. In a balances transportation model each of the constraints is an equation, and the problem could be formulated like that:

$$minimize \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} \cdot x_{ij}$$

$$Subject\ to \quad \sum_{j=1}^{n} x_{ij} = a_i \quad \forall i = 1, \dots, m$$

$$\sum_{i=1}^{m} x_{ij} = b_i \quad \forall j = 1, \dots, n$$

$$x_{ij} \geq 0 \quad \forall i \, \forall j$$

If the problem is not balanced, it will be easier to transform it in a balanced one. If the total supply is more than the total demand, a dummy demand point will be added. The dummy demand will be the exceed of supply and the cost associate to this new point will be zero, because it is not real. These shipments to the dummy demand point represent the supply capacity unused. They are therefore the slack variables of supply constraints or capacity.

If the total demand exceed the total supply, the problem is feasible.

3.1.3. Process Selection Problem

In this kind of problems, every product could be produced with different process. Sometimes the quantity of the products is fixed but other times is necessary to choose the quantity of each. The unit production cost depend on the process. Furthermore, there is some resources available in each period, which are used by the different process.

The problem is to know the quantity of each product are going to be produced in each process, manufacturing the total demand minimizing cost or maximizing benefits and considering the resources limits.

The elements in this problem will be:

$n =$ Number of products in the production process.

$m =$ Number of process that are available.

$d_j =$ Demand of product *j,* with $j = 1, \dots, n$

$b_i =$ Time available to the process i, with $i = 1, \dots, m$

$c_{ij} =$ Cost to produce one unit of the product j with the process i (∞ if it is not possible)

$t_{ij} =$ Time to manufacture one unit of the product j with the process i (0 if it is not available)

The decision variables will be:

x_{ij} = Quantity of product j produced in the process i.

The mathematical model to solve will be:

$$minimize \quad \sum_{j=1}^{n}\sum_{i=1}^{m} c_{ij} \cdot x_{ij}$$

$$Subject\ to \quad \sum_{i=1}^{m} x_{ij} = d_i \qquad \forall j = 1, \dots, n$$

$$\sum_{j=1}^{n} t_{ij} \cdot x_{ij} \leq b_i \qquad \forall i = 1, \dots, m$$

$$x_{ij} \geq 0 \qquad \forall j = 1, \dots, n$$

$$\forall i = 1, \dots, m$$

Other times, there is not specific demand to the production of each product and then, it is considered the sell prize v_j of the product j. In that case, the objective is maximize the benefit, and the model would be:

$$minimize \quad \sum_{j=1}^{n}\sum_{i=1}^{m} (v_j - c_{ij}) \cdot x_{ij}$$

$$Subject\ to \quad \sum_{j=1}^{n} t_{ij} \cdot x_{ij} \leq b_i \qquad \forall i = 1, \dots, m$$

$$x_{ij} \geq 0 \qquad \forall j = 1, \dots, n$$

$$\forall i = 1, \dots, m$$

3.1.4. Assignment Problem

The general assignment problem is the assignment of m labours to n machines minimizing the cost or total efficiency of assignment. In these kind of problems, the decision variables are $x_{ij} = 1$ if the labour i is assigned the machine j and $x_{ij} = 0$ if there is not assigned.

$$x_{ij} = \left\{ \begin{matrix} 1 & if\ the\ labour\ i\ is\ assigned\ the\ machine\ j \\ 0 & in\ other\ case \end{matrix} \right\}$$

The problem elements are:

m = Number of labours

n = Number of machines

c_{ij} = Cost or efficiency to assign the labour i to the machine j.

And the mathematical model are:

$$minimize \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} \cdot x_{ij}$$

$$Subject\ to \quad \sum_{j=1}^{n} x_{ij} \leq 1 \quad \forall i = 1, \dots, m$$

$$\sum_{i=1}^{m} x_{ij} \geq 1 \quad \forall j = 1, \dots, n$$

$$x_{ij} \in \{0,1\}$$

If the number of machines is less than the labours, it should be introduced a fictitious machine to balance the problem.

The assignment problem is a type of the transport problem, with integer constraints.

3.2. Solving Linear Programming Problems: The Simplex Method

The method used to solve LP problems is the Simplex Method, developed by George Dantzig in 1947. Beginning at the origin, this algorithm moves from one vertex of the feasible region to an adjacent vertex in such a way that the value of the objective function improvement or stays the same; it never gets worse. This movement continues until the vertex that yields the optimal solution is reached.

The steps of this method could be sumarize in:

- Transform the LP problem in the standard form.
- Create the simplex tableau.
- Make pivoting process.

All this is explained bellow

3.2.1. Standard Form of a Linear Programming Problem

Before starting with the Simplex Method, the LP problem must be in the standard form:

$$Maximize\ (z) = c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n \cdot x_n$$

$$Subject\ to \qquad a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$

$$\dots \qquad \dots \qquad \dots$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m$$

$$x_1 \geq 0,\ x_2 \geq 0, \ldots\ x_n \geq 0$$

where the objective is maximize, the constraints are equalities and the variables are all nonnegative.

The matrix in the standard form are:

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \quad c = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \quad A = \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \ldots & a_{mn} \end{pmatrix} \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

If the problem is not standard, it must be converted as follows:

- If the problem is $min\, z$, convert it into $\max -z$.
- If a constraist is $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \leq b_i$, convert it into an equality constraint by adding a nonnegative slack variable s_i. The resulting constraint is $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n + s_i = b_i$, where $s_i \geq 0$.
- If a constraist is $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \geq b_i$, convert it into an equality constraint by subtracting a nonnegative surplus variable s_i. The resulting constraint is $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n - s_i = b_i$, where $s_i \geq 0$.
- If some variable x_j is unrestricted in sign, replace it everywhere in the formulation by $x_j' - x_j''$, where $x_j' \geq 0$ and $x_j'' \geq 0$.

Example. Find the maximum value of $z = 4 \cdot x_1 + 6 \cdot x_2$*, where* $x_1 \geq 0$ *and* $x_2 \geq 0$*, subject to the following constrains:*

$-x_1 + x_2 \leq 11$

$x_1 + x_2 \leq 27$

$2 \cdot x_1 + 5 \cdot x_2 \leq 90$

The left-hand side of each inequality is less than or equal to the right-hand side. Slack variables must be added to the lef side of each equation to produce the following system of linear equations:

$-x_1 + x_2 \leq 11 \quad \rightarrow \quad -x_1 + x_2 + s_1 = 11$

$x_1 + x_2 \leq 27 \quad \rightarrow \quad x_1 + x_2 + s_2 = 27$

$2 \cdot x_1 + 5 \cdot x_2 \leq 90 \quad \rightarrow \quad 2 \cdot x_1 + 5 \cdot x_2 + s_3 = 90$

3.2.2. The Simplex Tableau

The simplex method is carried out by performing elementary row operations on a matrix that it is called the simplex tableau. This tableau consists of the augmented

matrix corresponding to the constraint equations together with the coefficients of the objective function cleared in the form:

$$z - c_1 \cdot x_1 - c_2 \cdot x_2 - \cdots - c_n \cdot x_n = 0$$

Example.

$$z - 4 \cdot x_1 - 6 \cdot x_2 = 0$$

In the columns of the tableau will be all the variables of the problem, included the slacks variables. In the rows will be the coefficients of the equalities obtained, one row for each restriction and the last row with the coefficients of the objective function.

Basic Variables	x_1	…	x_n	s_1	…	s_m	b
s_1	a_{11}	…	a_{1n}	1	…	0	b_1
…	…	…	…	…	…	…	
s_m	a_{m1}	…	a_{mn}	0	…	1	b_n
	$-c_1$	…	$-c_n$	0	0	0	

For this initial simplex tableau, the basic variables are $s_1, \ldots, s_m$, and the nonbasic variables are $x_1, \ldots, x_n$.

Example. Simplex tableau.

Basic Variables	x_1	x_2	s_1	s_2	s_3	b
s_1	-1	1	1	0	0	11
s_2	1	1	0	1	0	27
s_3	2	5	0	0	1	90
	-4	-6	0	0	0	

It could be seen that the current solution is: $(x_1, x_2, s_1, s_2, s_3) = (0,0,11,27,90)$.

To perform an optimality check for a solution represented by a simplex tableau, it is looked at the entries in the bottom row of the tableau. If any of these entries are negative, then the current solution is not optimal.

3.2.3. Pivoting

Once it is set up the initial simplex tableau for a linear programming problem, the simplex method consists of checking for optimality and then, if the current solution is not optimal, improving the current solution.

To improve the current solution, a new basic variables is brought into the solution (it is called the entering variable). This implies that one of the current basic variabes must leave, otherwise it would be many variables for a basic solution (called the departing variable).

The entering and departing variables are chosen as follows:

1) The entering variable corresponds to the smallest (the most negative) entry in the bottom row of the tableau.
2) The departing variable corresponds to the smallest nonnegative ratio of $\frac{b_i}{a_{ij}}$, in the column determined by the entering variable.
3) The entry in the simplex tableau in the entering variable's column and the departing variable's row is called the pivot

Finally, to form the improved solution, it is applied Gauss-Jordan elimination to the column that contains the pivot. This process is called pivoting.

Example. Pivoting

The current solution of the previous section corresponds to a z-value of 0. To improve this solution, we determine that x_2 is the entering variable, because -6 is the smallest entry in the bottom row.

Basic Variables	x_1	$\boldsymbol{x_2}$	s_1	s_2	s_3	b
s_1	-1	1	1	0	0	11
s_2	1	1	0	1	0	27
s_3	2	5	0	0	1	90
	-4	$\boldsymbol{-6}$	0	0	0	

To find the departing variable, we locate the b_i's that have corresponding positive elements in the entering variables column and form the following ratios:

$$\frac{11}{1} = 11 \quad \frac{27}{1} = 27 \quad \frac{90}{5} = 18$$

The smallest positive ratio is 11, so it is chosen s_1 as the departing variable.

Basic Variables	x_1	x_2	s_1	s_2	s_3	b
$\boldsymbol{s_1}$	-1	1	1	0	0	11
s_2	1	1	0	1	0	27
s_3	2	5	0	0	1	90
	-4	-6	0	0	0	

The pivot is the entry in the first row and the second column. After, it is used Gauss-Jordan elimination to obtain the following improved solution:

$$\begin{pmatrix} -1 & 1 & 1 & 0 & 0 & 11 \\ 1 & 1 & 0 & 1 & 0 & 27 \\ 2 & 5 & 0 & 0 & 1 & 90 \\ -4 & -6 & 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} -1 & 1 & 1 & 0 & 0 & 11 \\ 2 & 0 & -1 & 1 & 0 & 16 \\ 7 & 0 & -5 & 0 & 1 & 35 \\ -10 & 0 & 6 & 0 & 0 & 66 \end{pmatrix}$$

x_2 has replace s_1 in the basis column and the improved solution $(x_1, x_{2,} s_1, s_2, s_3) = (0,11,0,16,35)$ has a z-value of 66 ($z = 4 \cdot 0 + 6 \cdot 11 = 66$).

This process continues until it is reached the optimal solution.

3.2.4. Simplex method output

When a linear programming problem is solved by the Simplex Method, it is produced one of the following outputs:

- The problem has not feasible solutions (non-feasible problem). There is not set of decision variables values that verified all restrictions.
- The problem has finite optimal solution and it is found by the method. This is the normal situation and it is found the optimal values of the decision variables. Two cases could happened: there is only one optimal solution or there are infinity optimal solutions (alternative optimal solutions).
- The problem is not bounded. There is not a point where the optimum value is reached, if not there are set of points where the objective function are growing indefinitely (in a maximum problem) or decreasing indefinitely (minimum problem), and the maximum or minimum value of the objective function is $+\infty$ or $-\infty$ respectively. Although that case is possible mathematically, in practice it is unlikely to occur because generally the variables have bounds.

4. Computational complexity

The most limited resources in a software production are:

- The execution time
- The memory used to solve the problem

With these two variables is done the study of the computational complexity of the algorithms.

The complexity of the algorithms is a function of input size of the problem, n. The elementary steps number (T) that have to do the algorithm will be less equal than a constant (k) by the function that depends of the size of the problem:

$$T \leq k \cdot f(n)$$

So the algorithm has a complexity order of $f(n)$ and it is written as $O\big(f(n)\big)$.

Algorithms, depending of their complexity order, could be classified into two groups: polynomials o exponentials. A polynomial algorithm is one whose complexity function is $O(n^n)$ o less. It is called efficient algorithm. Algorithms for which it is not possible to limit the complexity of polynomial form are called exponential algorithms.

	Notation	Name
Polynomial	$O(1)$	Constant
	$O(\log(n))$	Logarithmic
	$O(n)$	Linear
	$O(nlog(n))$	Loglinear, Linearithmic, Quasilinear or Supralinear
	$O(n^2)$	Quadratic
	$O(n^3)$	Cubic
	$O(n^n)$	Polynomial
Exponential	$O(c^n)$	Exponential
	$O(n!)$	Factorial

4.1. Complexity class

Before starting with the complexity class of the problems, it is necessary to clarify the difference between a deterministic Turing Machine and a non-deterministic Turing Machine.

The Turing Machine is a hypothetical universal computing machine able to modify its original instructions by reading, erasing or writing a new symbol on a moving tape of fixed length that acts as its program.

In a deterministic Turing Machine, for each pair (state, symbol), there is at most a transition to another state. However, in a non-deterministic Turing Machine exists at least one pair (state, symbol) over a transition to different states.

After this clarification, the class of the problems could be P, NP or NP-hard.

P class (Polynomial-time) contains the decision problems that a deterministic Turing Machine can solve in a polynomial time. These problems are treatable because they could be solved in a reasonable time. Most common problems (sorting, searching…) belong to this class.

NP class (Non-Deterministic Polynomial-time) has the decision problems that a non-deterministic Turing Machine can solve in polynomial time. As every deterministic Turing Machine is a particular case of a non-deterministic Turing Maching, each P problem is too an NP problem: $P \subseteq NP$. But know if $P = NP$ or $P \neq NP$ is a problem without solution nowadays.

NP hard problems are NP problems whose complexity is extreme. These problems are equivalents between each others. It is believed that there are not polynomial algorithms to solve them (unproved). The travel salesman problem or knapsack problem are an example of these kind of problems.

5. SCIP Optimization Suite

The SCIP Optimization Suite is a toolbox for generating and solving mixed integer programs. It consists of the following parts [8]:

- SCIP is the mixed integer (linear and nonlinear) programming solver and constraints programming framework.
- SoPlex is the linear programming solver.
- ZIMPL is the mathematical programming language.
- UG is parallel framework for mixed integer (linear and nonlinear programs).
- GCG is the generic branch-cut-and-price solver.

It can be easily generate linear programs and mixed integer programs with the modelling language ZIMPL. The resulting model can directly be loaded into SCIP and solved. In the solution process, SCIP may use SoPlex as underlying LP solver [8].

In this research, it is only used SCIP and SoPlex, and it is not used the ZIMPL language. Instead, it is used C/C++ and other formats as LP or MPS as it is described later.

5.1. Installation

The first thing is to know the operating system of the computer. If it is LINUX, the installation process would be easier, but if it is Windows, we should install in our computer an emulation of Linux. It must be Cygwin or MinGW.

This research is done under Windows operating system. For that reason, it is decided to install Cygwin. In the following link (https://cygwin.com/install.html) is possible to download it (32 bits or 64 bits version). From here only it will be exposed this alternative.

After the download, it is followed the next steps:

1) Install from the internet.
2) Write the root directory.
3) Write the local package directory.
4) Choose Direct Connection.
5) Choose any available download sites.
6) Select the packages. It is mandatory:
   - Devel
   - Debug
   - KDE
   - Shell
   - Systems
7) Choose continue and wait until the program is installed.

Then, it must be downloaded the source code of SCIP Optimization Suite. It is needed a license, free for research proposes.

It is download in a .tgz format. First, that file should be in the local package directory (by default “home”). The steps will be:

1) Enter in the local package directory
2) Extract the package
   E.g.: tar xvf scipoptsuite-3.2.1.tgz
3) In the file extracted, enter make, with the options that it will be needed. In the installation file is explained all the parameters that it could be set up and which ones are by default (http://scip.zib.de/doc/html/INSTALL.php).

   In this research was set up:

   ```
   make test LPS=spx ZLIB=false ZIMPL=false READLINE=false
   ```

   LPS is the LP solver that it is wanted to use. In that case, it was chosen SoPlex (spx). ZLIB=false disable ZLIB usage, ZIMPL=false disable ZIMPL file reader and READLINE=false disable readline library, because it was not needed.

After that, SCIP Optimization Suite will be installed.

6. SoPlex

6.1. Introduction

SoPlex is an optimization package for solving linear programming problems (LPs) based on an advanced implementation of the primal and dual revised simplex algorithm. It provides special support for the exact solution of LPs with rational input data. It can be used as a standalone solver reading MPS or LP format files via a command line interface as well as embedded into other programs via a C++ class library [9].

SoPlex has been used in numerous researches and industry projects and is the standard LP solver linked to the constraint integer-programming solver SCIP [8][9].

6.2. Installation

Once installed SCIP Optimization Suite, it is entered in the directory of that and it could be seen a soplex-version.tgz format. The steps will be:

1) Extract the package
   E.g.: tar xvf soplex-2.2.1.tgz
2) Inside the directory of SoPlex, make test with the options that it will be needed. In the installation file is defined all the parameters (http://soplex.zib.de/doc/html/INSTALL.php).
   In this research, it was entered:
   ```
   make test ZLIB=false ZIMPL=false READLINE=false
   ```

6.3. SoPlex Operation

As previously mentioned, SoPlex could solve MPS or LP format files by a command line interface or embedded programs by a C++ library.

This research is focus on solve by a command line interface, for that reason hereafter it is exposed only this alternative.

First, it is needed writing the problem in a LP or MPS format. As following, it is explained both and in the section 6.3.3., some examples are written in these formats, but to study the performance of SoPlex, it is chosen the LP format because is more intuitive and easier than the MPS format, as it could be seen in the following sections.

6.3.1. Formats

6.3.1.1. LP format

The lp-format is lpsolves native format to read and write lp models [10].

The lp-format input syntax is a set of algebraic expressions and "int" declarations in the following order:

1) Objective function

2) Constraints
3) Bounds
4) Declaration
5) End

The objective function is a linear combination of optional variables and constants, ending with a semicolon. To indicate whether you want it to be minimized or maximized, this section beginning with one of the keywords:

- max
- maximun
- maximize
- min
- minimun
- minimize

Maximization is the default.

The objective may be given a name by writing before the expressions. If no name is given, then the objective is name “obj”.

The constraints must begin with one of the keywords:

- subj to
- subject to
- s.t.
- st

A constraint contains an optional name followed by a colon plus a linear combination of variables and constants followed by a bound type (<,<=,=,>,>=) and the bound may be a number. There is no semantic difference between “<” and “<=” or between “>” and “>=”.

Bounds on the variables can be specified in the bound section beginning with one of the keywords:

- bound
- bounds

The bounds section is optional but should, if present, follow the “subject to” section. All the variables listed in the bounds section must occur in either the objective or a contstraint. Each bound definition must begin in a new line.

The default lower and upper bounds are 0 and $+\infty$. A variable may be declared free with the keywork “free”, which means that the lower bound is $-\infty$ and the upper bound is $+\infty$. Furthermore it may be assigned a finite lower and upper bound. The bound definitions for a given variable may be written in one or two lines, and bounds can be any number or $\pm\infty$ (written as +inf/-inf/+infinity/-infinity).

The declaration section is optional and it could be to define variables as binary or define general variables.

The keywords for the first type could be:

- bin
- binaries
- binary

And for the second type:

- gen
- general

Again, all variables listed in the binary or general sections must occur in either the objective or a constraint.

Finally, and LP-format file must end with the keyword:

- end

Examples of LP format are written in the section 6.3.3.

6.3.1.2. MPS format

The main things to know about fixed MPS format are that it is column oriented (as opposed to entering the model as equations), and everything (variables,rows, ect.) gets a name [10].

MPS is an old format, so it is set up as though you were using punch cards. Fields start in column 2, 5, 15, 25, 40 and 50. Sections of an MPS file are marked by so-called header cards, which are distinguished by their starting in column 1. The names that are chosen for the individual entities (constraints or variables) are not important to the solver.

Following there is a guide to know how to write a MPS file:

Field:	1	2	3	4	5	6
Columns:	2-3	5-12	15-22	25-36	40-47	50-51
	NAME		Prob_name			
	ROWS					
	type	name				
	RHS					
		rhs name	row name	value	row name	value
	COLUMS					
		column name	row	value	row name	value
	BOUNDS					
	type	bound name	column name	value		
	END DATA					

There is nothing in MPS format that specifies the direction of optimization. As default, most MPS codes minimize (SoPlex minimizes as default). If it is wanted to maximize, it must be written between the NAME section and the ROWS section the following:

```
OBJSENSE
 MAX
```

The name record can have any value, starting in column 15.

The ROW section defines the names of all the constraints; entries in column 2 or 3. The code for indicating row type is as follows:

Type	Meaning
E	Equality
L	Less than or equal
G	Greater than or equal
N	Objective
N	No restriction

The COLUMNS section contains the entries of the A-matrix. All entries for a given column must be placed consecutively, although within a column the order of the entries (rows) is irrelevant. Rows not mentioned for a column are implied to have a coefficient of zero.

The RHS section allows one or more right-hand-side vectors to be defined; there is seldom more than one.

The optional BOUNDS section is where the bound of variables are specified. When bounds are not indicated, the default bounds ($0 \leq x < \infty$) are assumed. The code for indicating bound type is as follows:

Type	Meaning
LO	lower bound b <= x (< +inf)
UP	upper bound (0 <=) x <= b
FX	fixed variable x = b
FR	free variable -inf < x < +inf
MI	lower bound -inf -inf < x (<= 0)
PL	upper bound +inf (0 <=) x < +inf
BV	binary variable x = 0 or 1
LI	integer variable b <= x (< +inf)

UI	integer variable (0 <=) x <= b
SC	semi-cont variable x = 0 or l <= x <= b l is the lower bound on the variable If none set then defaults to 1

Another optional section called RANGES specifies double-inequalities. Ways to mark integer variables are also beyond (keywork MARKER and possibly SOS are involved).

The final card must be ENDATA (notice the odd spelling).

Examples of MPS format are written in the section 6.3.3.

6.3.2. Solving a problem with SoPlex

To solve the LP or MPS file, it must be in the bin directory inside the directory of SoPlex directory, because is where it was installed.

E.g.: C:\cygwin64\home\scipoptsuite-3.2.1\soplex-2.2.1\bin

With the emulation of Linux (Cygwin) it is needed to access to this directory too and to solve the problem it should be enter the following command line:

./soplex “Nameofthefile”.lp
./soplex “Nameofthefile”.mps

The result that appear in the display is, in general, the number of iterations and the objective value. However, if it is required other information, it could be entered in the command line other options.

The general options are:

`--readbas=<basfile>`	Read starting basis form file
`--writebas=<basfile>`	Write terminal basis to file
`--writefile=<lpfile>`	Write LP to file in LP or MPS format depending on the extension
`--<type>:<name>=<val>`	Change parameter value using syntax or settings file entries
`--loadset=<setfile>`	Load parameters from settings file (overruled by command line parameters)
`--saveset=<setfile>`	Save parameters to settings file
`--diffset=<setfile>`	Save modified parameters to settings file

To set limits and tolerances:

-t<s>	Set time limit to <s> seconds
-i<n>	Set iteration limit to <n>
-f<eps>	Set primal feasibility tolerance to <eps>
-o<eps>	Set dual feasibility (optimality) tolerance to <eps>

To set algorithmic settings:

--readmode=<value>	Choose reading mode for <lpfile> (0* - floating-point, 1 - rational)
--solvemode=<value>	Choose solving mode (0 - floating-point solve, 1* - auto, 2 - force iterative refinement)
-s<value>	Choose simplifier/presolver (0 - off, 1* - auto)
-g<value>	Choose scaling (0 - off, 1 - uni-equilibrium, 2* - bi-equilibrium, 3 - geometric, 4 - iterated geometric)
-p<value>	Choose pricing (0* - auto, 1 - dantzig, 2 - parmult, 3 - devex, 4 - quicksteep, 5 - steep)
-r<value>	Choose ratio tester (0 - textbook, 1 - harris, 2 - fast, 3* - boundflipping)

(*indicates default)

And the display options are:

-v<level>	set verbosity to <level> (0 - error, 3 - normal, 5 - high)
-x	Print primal solution
-y	Print dual multipliers
-X	Print primal solution in rational numbers
-Y	Print dual multipliers in rational numbers
-q	Display detailed statistics
-c	Perform final check of optimal solution in original problem

6.3.3. Linear problems with SoPlex

6.3.3.1. Example 1

A company manufactures desks, tables and chairs. The manufacture of each type of wood and furniture requires two types of skilled labor: carpentry and finishing. The amounts of each resource needed to produce each type of furniture and other details of the problem are given in the table below (the amounts of wood are measured in feet - table)

Resource	Desk	Table	Chair	Resource quantity
Wood	8	6	1	48
Finishing hours	4	2	1,5	20
Carpentry hours	2	1,5	0,5	8
Sales prize	60 €	30 €	20 €	
Minimum demand	3	1	0	
Maximum demand	No limits	5	No limits	

Assuming that the available resources have already been purchased, find an optimal solution.

Solution

This is a product-mix problem. There is three products, which are going to be the variables, and the objective is maximize the benefit.

x1 = “Number of desks”

x2 = “Number of tables”

x3 = “Number of chairs”

First, it is necessary writing the LP file (in the appendix, section A.1.1.).

Later, this file must be in the bin directory. In addition to the objective value, it is wanted to know the quantity of the variables. For that reason, in the command line it is added “-x”.

```
./soplex –x Example1.lp
```

The display on screen is written in section A.2.1. of the appendix.

The objective value is 230€ and the value of the variables are:

- x1 (“Number of desk”) = 3 units
- x2 (“Number of tables”) = 1 unit
- x3 (“Number of chairs”) = 1 unit

The iterations number is two and the compilation time is approx. 0 seconds.

This problem could be done in a MPS format too, as in the section A.2.2.

To solve it, it is written in the command line:

```
./soplex –x example1.mps
```

And the result is the same as solving the problem with the LP format.

This problem was implemented in Xpress Mosel (another program to Optimization problems) to be sure that the problem solving with SoPlex was correctly done, and it gives the same result.

6.3.3.2. Example 2

A company has three factories generating electricity that supply to four cities. Each factory can supply a limited amount of energy and should satisfy the requirements of all cities. The following table lists the maximum amounts of energy that can generate each plant, demand in each city (both data given in millions of kw/h) and costs in €, sending one million kwh from each plant to each city shown. It is wanted to know the distribution of energy that minimize the total cost.

	City 1	City 2	City 3	City 4	Capacity
Factory 1	8	6	10	9	35
Factory 2	9	12	13	7	50
Factory 3	14	9	16	5	40
Demand	45	20	30	30	

Solution

This is a transport problem. There are 3 sources and 4 destinations, so in total there are 12 decision variables, and the objective is minimize the cost to distribute the energy.

First, it is written the LP file (in the appendix, section A.1.3.).

And later, it is solved with SoPlex and the result is written in section A.2.2.

The objective function value is 1020 €, and variables values are:

$$x_{11} = 0 \qquad x_{12} = 10 \qquad x_{13} = 25 \qquad x_{14} = 0$$

$$x_{21} = 45 \qquad x_{22} = 0 \qquad x_{23} = 5 \qquad x_{24} = 0$$

$$x_{31} = 0 \qquad x_{32} = 10 \qquad x_{33} = 0 \qquad x_{34} = 30$$

With an MPS format, the result would be the same, but it is not implemented with that because MPS format is longer and less intuitive than the LP format.

6.4. SoPlex performance

6.4.1. Introduction

This research objective is evaluated the complexity of SoPlex. Then it is necessary to find a way to create big problems with, at least, one feasible solution.

It is used the C language to program and generate the problems. First, to each kind of problems, an LP file is generated under a known problem to test if the writing and format problem would be the correct when SoPlex tried to solve it. Later, the C program is changed to generate problems with random data, which a feasible solution.

Chosen problems are Product-Mix and Transportation Problem. As follows it is explained the steps to generate the problem and it is analysed the results to evaluate effectiveness of SoPlex.

6.4.2. Product-Mix Problem

6.4.2.1. Program

The first step has been done is create a LP file from a known problem with C language. The Product-Mix mathematical problem, which has been exposed in the chapter 2, is:

$$maximize \quad \sum_{j=1}^{n} (p_j - c_j) \cdot x_j$$

$$Subject\ to \quad \sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i \qquad \forall i = 1, \dots, m$$

$$x_j \leq u_j \qquad \forall j = 1, \dots, n$$

$$x_j \geq l_j \qquad \forall j = 1, \dots, n$$

And the problem has been chosen to try the problem with SoPlex is:

> *Example: It is going to be process 4 products in 3 different machines. Time (in minutes) required per unit of each product , the ability of daily operation of each machine in minutes and quantities to produce minimum and maximum for each product selling prices and costs per unit produced each product are in the following table:*

	Product 1	Product 2	Product 3	Product 4	Capacity
Machine 1	1	1.5	2	1	450
Machine 2	3	2	0	2	400
Machine 3	2	0.5	3	1	620
Q_min	2	20	30	30	
Q_max	200	200	150	100	
Prize/unit	25	20	30	15	
Cost/unit	10	15	18	5	

First it is defined in the C program the variables which will be used. The C program is in the appendix (A.3.1.).

In the program, it is introduced the values by hand. Later, it is calculated the profit margin, which is prize minus cost:

$$v_j = p_j - c_j \quad \forall j$$

Once it is defined and calculated all the parameters, it is proceed to generate the LP file. It must be written the next line to create it:

```
freopen("ProductMixResult.lp","w",stdout);
```

and, when all the file has been written, it must be closed:

```
fclose(stdout);
```

In the middle, all the characters inside a “printf” are written in the LP file.

When the program is compiled, the LP file generated is in the appendix (Section A.1.4.).

After that, it is tested that the LP file has the correct format running SoPlex, and the result is in section A.2.3.

The value of the objective function, that is the benefit, is 3.508 € and the number of each product are:

Product 1 $(x_1) = 56$

Product 2 $(x_2) = 20$

Product 3 $(x_3) = 134$

Product 4 $(x_4) = 96$

Once the format has been tested, it will be used in the following steps the part of the program that generate the LP file. The other part will be changed with the purpose to generate an automatically Product-Mix problem.

To generate automatically a random problem with feasible solution, it is generated first the solution (x), and later the rest of the values. Then it is explained step by step the procedure.

First, in an external file has been defined the number of products (n), number of resources (m) and the deviation for the lower and upper bound (later it will be explained). The program is written in section A.3.2.

After, it is define the matrix's and vector's problem. It must be taken into account that the propose of these research is generate big problems. For that reason, the matrix and vectors must be defined in the dynamic way, using pointers.

As the problem will be generated by random, there is two possibilities: generating from a seed or from the computer time. The first option is only enter a number by hand:

```
seed = 4853;
srand(seed);
```

With the second option, at the beginning of the program, outside the main function, it must be included "time.h" and, to generate the random numbers, the date and the time of the system is taken.

```
#include <time.h>

int main()
{
    …
    srand(time(NULL));
    …
}
```

The chosen option is the first, because it is the only way to repeat a simulation with the same values.

The principal problem is generated the capacity of each resource ensuring that there is, at least, one feasible solution. For that reason, it is generated first the a_{ij} matrix and the random solution $(\tilde{x}_j)$. With these values is calculated the capacity of each resource (b_i):

$$b_i = \sum_{j=1}^{n} a_{ij} \cdot \tilde{x}_j + rand\_number \qquad \forall i = 1, \dots, m$$

The values of product time in each resource (a_{ij}) will be between 0 and 10 and the values of the random solution $(\tilde{x}_j)$ between 0 and 100, all of them integers. To generate the capacity vector, it is calculated every constraint and later it is added a random number.

Later, the bounds are calculated. The upper bounds will be the random solution plus de upper deviation defined in the external file. The lower bounds will be the random solution less the lower deviation. If the lower bound calculation is negative, it will be written zero.

$$u_j = \tilde{x}_j + u\_deviation \qquad \forall j = 1, \dots, n$$

$$\begin{Bmatrix} l_j = \tilde{x}_j - l_{deviation} & if\ \tilde{x}_j - l_{deviation} > 0 \\ l_j = 0 & if\ \tilde{x}_j - l_{deviation} \leq 0 \end{Bmatrix} \quad \forall j = 1, \dots, n$$

Then, the prize and cost of each resource is generated by random, and the profit margin is calculated $(v_j = p_j - c_j)$.

Now, it could be created a random Product Mix Problem and the LP file to solve with SoPlex the problem. It is entered:

- n=4
- m=3
- u_desviation=200
- l_desviation=100
- Seed=4853

And the LP file has been generated is in the section A.1.5. To check if the format is correct, with solve the LP file with SoPlex, and the result is in section A.2.4.

6.4.2.2. Testing and results

Once it is proved that the C programs works correctly, it is made the study of SoPlex complexity.

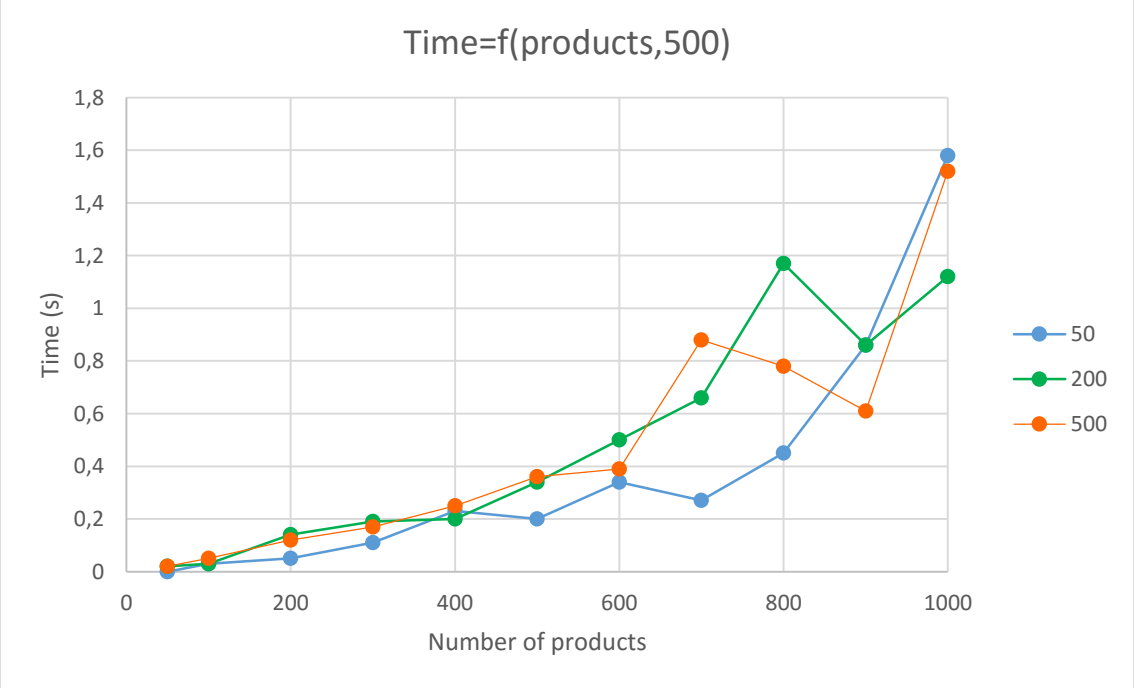
It is generated big problems, and it is studied the execution time and the memory used. The execution time is obtained by display on screen, when SoPlex is executed and gives the results. However, to know the memory used it has been used the resource monitor of windows. It is taken the memory used before running SoPlex and the peak while is running. It is calculated the difference, that will be the memory used.

During the execution of the program with difference size of products and machines, it was discovered that the lines of the LP file must not have more than 8190 characters. At first, that was a problem, because the variables could not be more than 1000 (products number in the product-mix problem). However, the number of resources do not have limit, because one resource add one row in the LP format and it does not modified the number of characters in the rows.

In this research, the execution time and the memory used are represented in a graph in function of the number of products or resources.

Before starting to generate random product-mix problems to evaluate the execution time and the memory used, it is checked if the parameters of the quantity deviation have influence or not. It is chosen a number of 500 resources (m=500) and it is running the program with different number of products and deviations of 50, 200 and 500 (appendix, section A.4.1.).

Later, with the data, it is done the following graph:



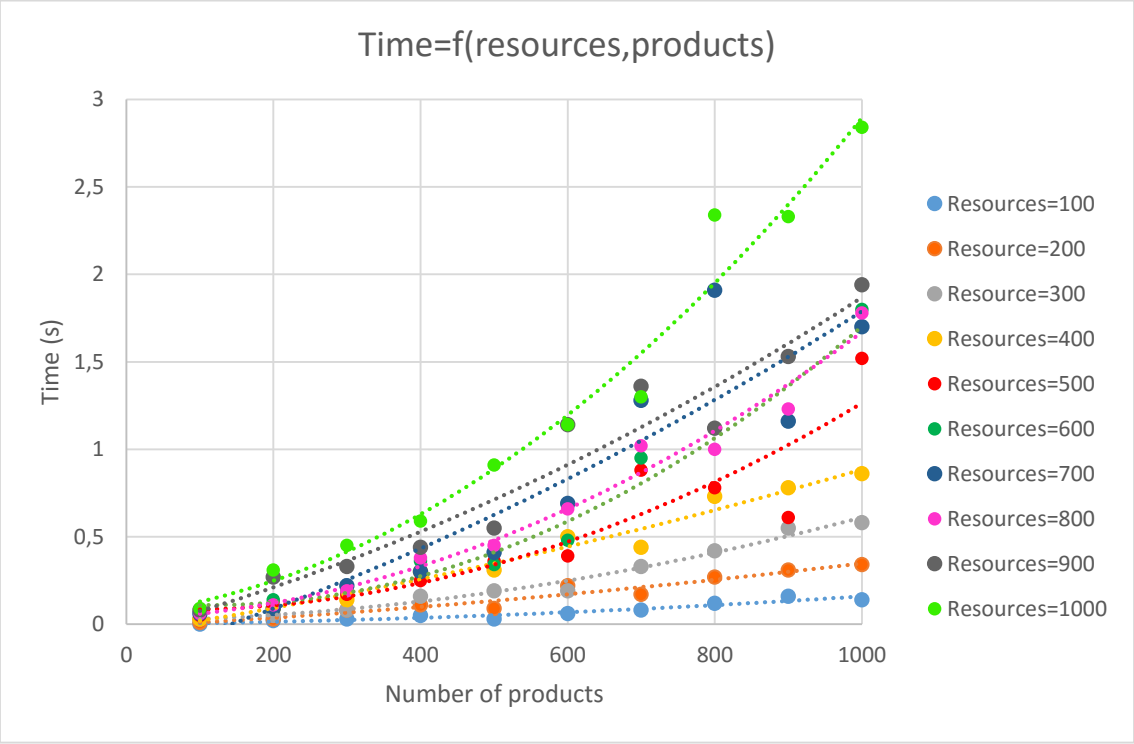
Graphical representation of the time as a function of products and the deviation parameters.

At the begging, it seems that with a lower deviation (50), the execution time is less, but it is random and depends of the random numbers that the program generates.

For that reason, it is concluded that the deviation parameter are not going to influence in a significant way in the research.

It is started to run the program, to $m = 100, 200, \ldots, 1000$ and $n = 100, 200, \ldots, 1000$, because we only can execute until $n = 1000$. The data obtained are in section A.4.2.

And with these data, it is created a graph that shows the evolution of the execution time depending on the resources (m) and the products (n):



Graphical representation of the time as a function of resources and products.

How it could be seen, when the number of resources and products grow, the time grows in a polynomial way.

To know the order of the polynomial dependence, it is used the excel tool of the trend line and the R^2, which says how good is the regression. In the following table, it can be seen the R^2 for each resource, with a polynomial order of 2 and 3:

Resources	**R^2 (Order=2)**	**R^2 (Order=3)**	**Difference**
100	0,922	0,9232	0,0012
200	0,9453	0,9453	0
300	0,9793	0,9796	0,0003
400	0,9679	0,9685	0,0006
500	0,8295	0,8414	0,0119
600	0,9655	0,9667	0,0012
700	0,8357	0,8779	0,0422
800	0,9749	0,9763	0,0014
900	0,9395	0,9398	0,0003

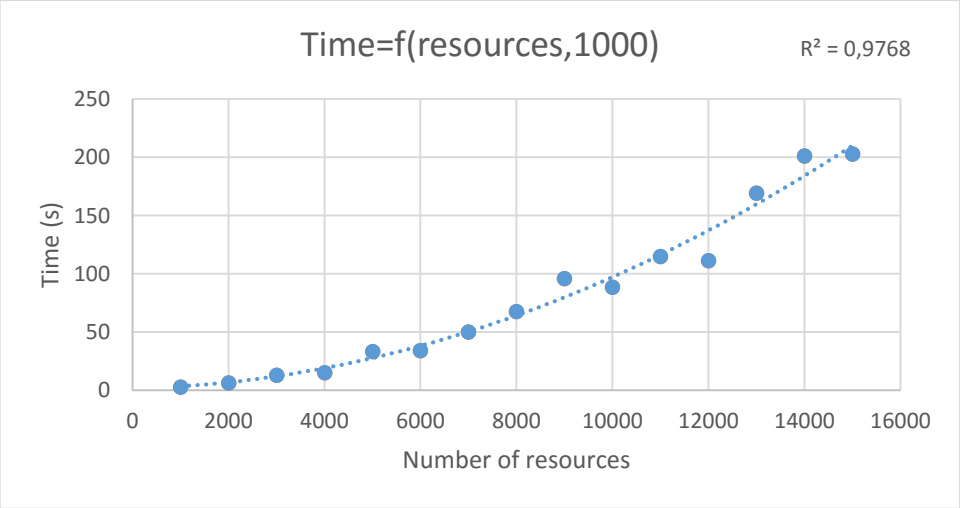
100	0,972	0,9722	0,0002

As it can be seen, the difference is very small between a polynomial regression of order 2 and 3. For that reason, the order of the function is 2 and the time grows in a polynomial way of order two $\left(O(n^2)\right)$.

The memory used is not represented in a graph because the values obtained are very small and are not significant.

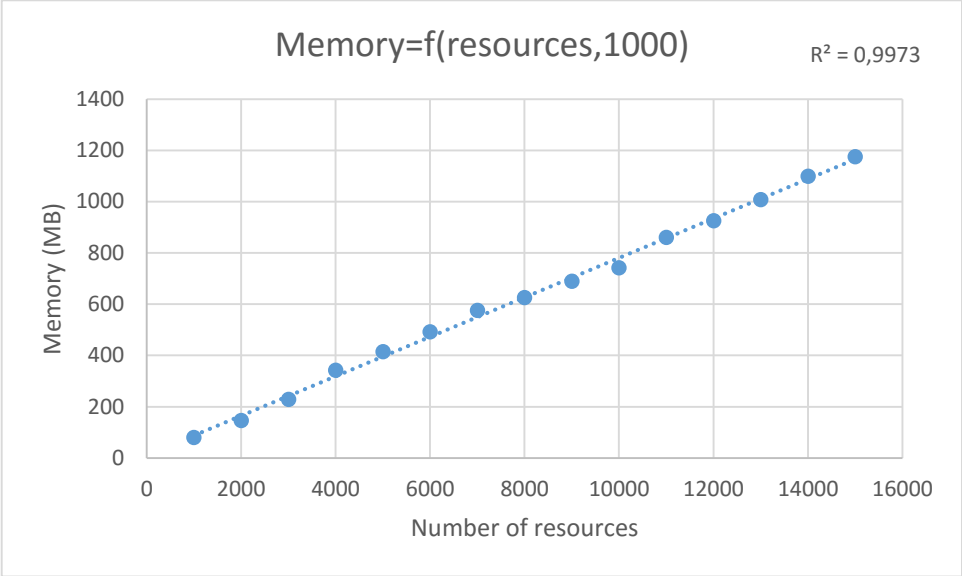
After, the number of products is set at 1000 (the maximum that can) and the evolution of the execution time and the memory used are studied increasing the resources number. It is run the program increasing the number the resources by thousands (section A.4.3.).

It is represented the execution time in a graph, function of the resources number. As it could be seen, the execution time grows in a polynomial form of order 2 with a R^2=0.9768.



Graphical representation of the time as a function of resources (by thousands)

Furthermore, it is represented too the memory used function of the resources number, because now the data are big enough to be representative. The memory grows in a linear form when the resources are increased $(O(n))$.

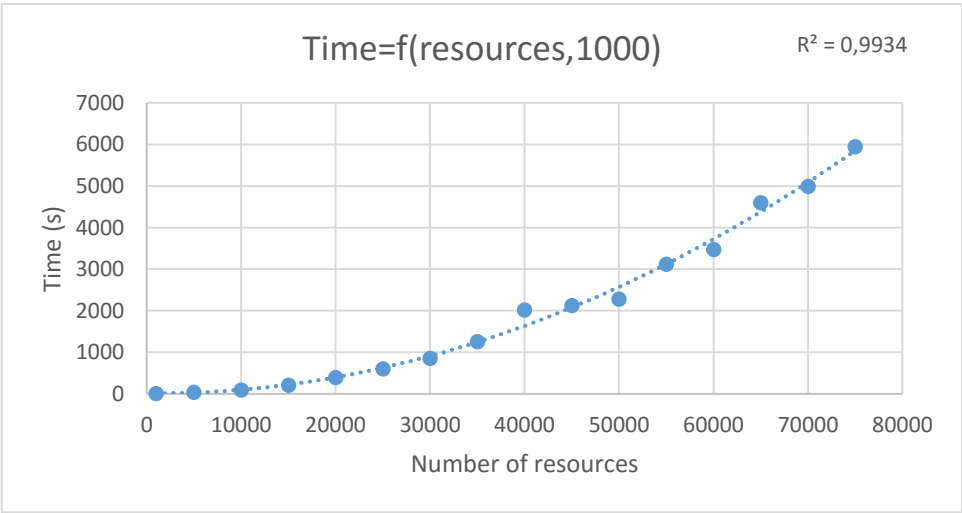


Graphical representation of the memory as a function of resources (by thousands)

After that, it is started to run the program by 5 thousand (section A.4.4), to reach the maximum resources that the computer, where SoPlex had been installed, could run. The computer has 8 GB of RAM memory and the processor is an intel core i5.

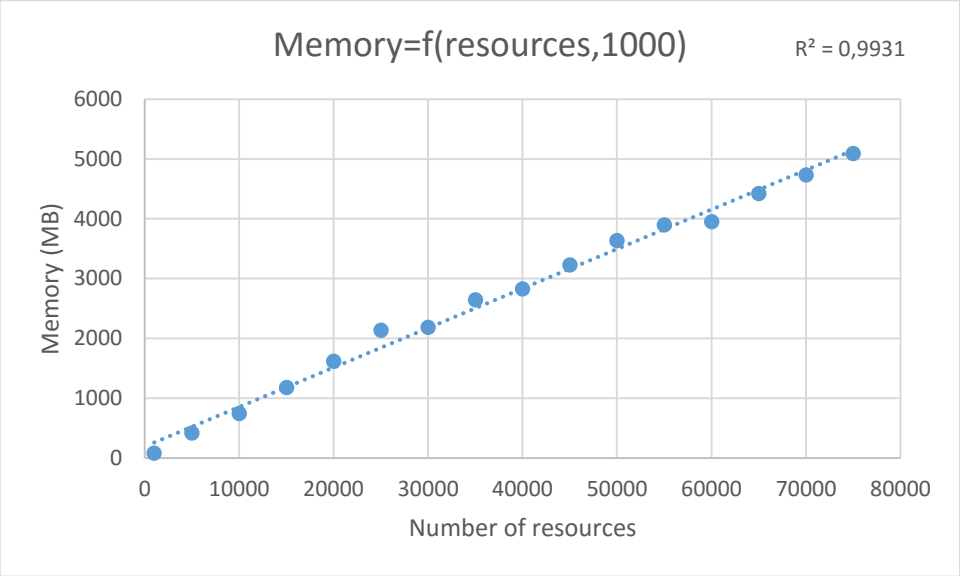
As it could be seen, it was stopped to execute the program when the memory where near 8 GB and the execution time was around 2 hours.

As following, it is represented the execution time it is confirmed again the polynomial dependence of order 2 (R^2=0.9934) with the resources number:



Graphical representation of the time as a function of resources (by 5 thousands)

The memory used is represented too, and it has a linear dependence with the resource number:



Graphical representation of the memory as a function of resources (by 5 thousands)

If the research was done with another computer more powerful, the number of resources, that the program would be run, would be more.

With this research, it is demonstrated that the Product-Mix problem belongs to the P class problems, so it is treatable and could be solved in a reasonable time.

6.4.3. Transport Problem

6.4.3.1. Program

First, it is created a C program of a known transport problem to create the format of the LP file (section A.3.3). The problem chosen is the Example 2 of the section 6.3.3.2.

It is defined the variables, entering by hand and later it is generated the LP file with the C program.

The normal way to write the decision variables will be `x[i][j]`, but to use less characters because of the 8190 characters problem, it is written as `xi_j`.

Once it is tested that the LP format generated by the C program is correct, it is written the random generator program.

To generate automatically a random problem with feasible solution, it is generated first the solution (x), and later the rest of the values. Then it is explained step by step the procedure and the program is in the appendix (A.3.4.).

First, in an external file has been defined the number of sources (m), number of destinations (n) and the capacity and demand deviation. So, it is necessary to enter these values in the program.

Later, matrix's and vector's problem's are generated in a dynamic way, using pointers.

As the problem is generated by random, we should introduce one of the two options explain in the Product-Mix problem program (Section 6.4.2.1.). It is chosen the second one.

Then, it is started to create the values of the variables. First, the random solution ($\hat{x}_{ij}$). It is fixed a maximum value of 50, but it could be whatever. Later, the cost matrix (c_{ij}), with a maximum value of 20 (it could be whatever too).

To generate the demand vector, for each value of the vector, it is add up all the variables $\hat{x}_{ij}$ that arrive to the destination j, and it is subtracted a random number that, at most, has the value of the demand deviation introduced by hand:

$$\sum_{i=1}^{m} x_{ij} = b_i - \min(rand, deviation_{demand}) \quad \forall j = 1, \dots, n$$

To generate the capacity vector it is following the same procedure: it is add up all the variables $\hat{x}_{ij}$ that leave the source i, and it is added a random number that, at most, has the value of the capacity deviation introduced by hand:

$$\sum_{j=1}^{n} x_{ij} = a_i + \min(rand, deviation_{capacity}) \quad \forall i = 1, \dots, m$$

To check the program, it is introduced the following values:

- n=4
- m=3
- demand_deviation=10
- capacity_deviation=20

The LP file generated is in section A.1.6. and the solution with SoPlex is in section A.2.5. of the appendiz.

6.4.3.2. Testing and results

It is executed the program with a large number of sources and destinations, but it is found a problem: the line size must not be more than 8190 characters. For the transport problem is a problem, because the objective function is the biggest line, and it could not be generated as the Product-Mix Problem.

It is execute the program with different numbers of sources and destinations, and the result is in the section A.4.5.

As maximum, the problem could have around 850 variables with the LP format, to SoPlex could solved it.

For that reason, it could not be done a complexity study of the Transport Problem writing in a LP format. However, with an MPS format it could be done, because the line of the file are not going to be more than 100 characters due to the column orientation of the file.

7. SCIP

SCIP is a solver for mixed integer programming (MIP) and mixed integer nonlinear programming (MINLP). It is also a framework for constraint integer programming and branch-cut-and-price [8].

A similar technique is used for solving both Integer Programs and Constraints: the problem is successively divided into smaller subproblems (branching) that are solved recursively.

Integer Programming and Constraint Programming have different strengths: Integer Programming have different strengths: Integer Programming uses LP relaxations and cutting planes to provide strong dual bounds, while Constraint Programming can handle arbitrary (non-linear) constraints and uses propagation to tighten domains of variables [8].

SCIP is implemented as C callable library and provides C++ wrapper classes for user plugins. It can also be used as a standalone program to solve mixed integer programs given in MPS, LP, flatzinc, CNF, OPB, WBO, PIP, or CIP format. Besides that SCIP can directly read ZIMPL models [8].

In this project, only an introduction of SCIP is done using one of the examples of the library (the Queens Problem). With this problem the complexity of SCIP is evaluated in a basic way.

7.1. Installation

After the installation of SCIP Optimization Suite, SCIP installation is similar to SoPlex installation. It is entered in the directory of SCIP Optimization Suite, and it is seen a scip-version.tgz. The steps are:

1) Extract the package
   E.g.: tar xvf scip-3.2.1.tgz
2) Inside the directory of SCIP, make test with the options that it will be needed. In the installation file is defined all the parameters.
   In this research, it was entered:

   ```
   make test ZLIB=false ZIMPL=false READLINE=false
   ```

7.2. The Queens Problem

7.2.1. Mathematical Model

The n-Queens problem is based to find a n-queens distribution in a nxn chessboard without the queens do not attack each other's. Therefore, it could not be found two Queens in the same row, column or diagonal.

This problem has two versions. The most simple is to find exactly on feasible solution for n value. The other version, more difficult, is to find all the feasible solutions.

This can be modeled as a binary program in the following way: let $x_{ij} \in \{0,1\}$ denote whether a queen is placed on the i row and the j column of the chessboard. Since the problem is to find a placement, the objective function is irrelevant. It is add, however, the redundat objective to maximize the number of placed qeens [11]:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}$$

Now, it is forced exactly one queen to be placed in every column and every row:

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, \dots, n$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, \dots, n$$

The diagonal rows are a little bit more complicated to write up:

$$\sum_{i=1}^{n-j+1} x_{i,j+i-1} \leq 1 \quad j = 1, \dots, n$$

$$\sum_{j=1}^{n-i+1} x_{i+j-1,j} \leq 1 \quad i = 1, \dots, n$$

$$\sum_{i=1}^{n-j+1} x_{i,n-j-i+2} \leq 1 \quad j = 1, \dots, n$$

$$\sum_{j=1}^{n-i+1} x_{i+j-1,n-j+1} \leq 1 \quad i = 1, \dots, n$$

This model is implemented in the examples with the installation os SCIP. In the following section is explained how to use and run it.

7.2.2. Testing and results

The Queens problem is one of the examples that SCIP include. When SCIP Optimization Suite is installed, it should be entered in the SCIP directory first, later in the examples and, at the end, to the Queens problem:

```
/scipoptsuite-3.2.1/scip-3.2.1/examples/Queens
```

In this directory, to run the Queens problem the first time, it is introduced in Cygwin command line:

```
make test ZLIB=false ZIMPL=false READLINE=false
```

It is entered the other parameters, as in the SoPlex case, because the SCIP Optimization suite was installed with that.

The test is done for n=1,2,4,8,16 (display on screen in section A.5.1.). That is define in the Makefile, so if it is wanted to change the number of Queens, it could be modified the Makefile and written the dimension of the chessboard (n).

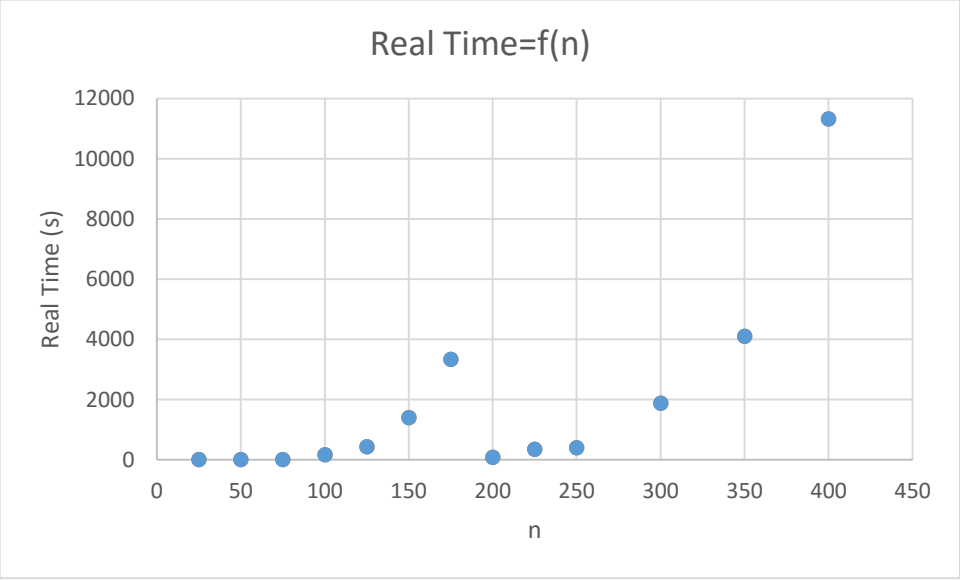
It is wanted to test the execution time of each program depending on n. For that reason, in the command line is included the command "time":

```
time make test ZLIB=false ZIMPL=false READLINE=false
```

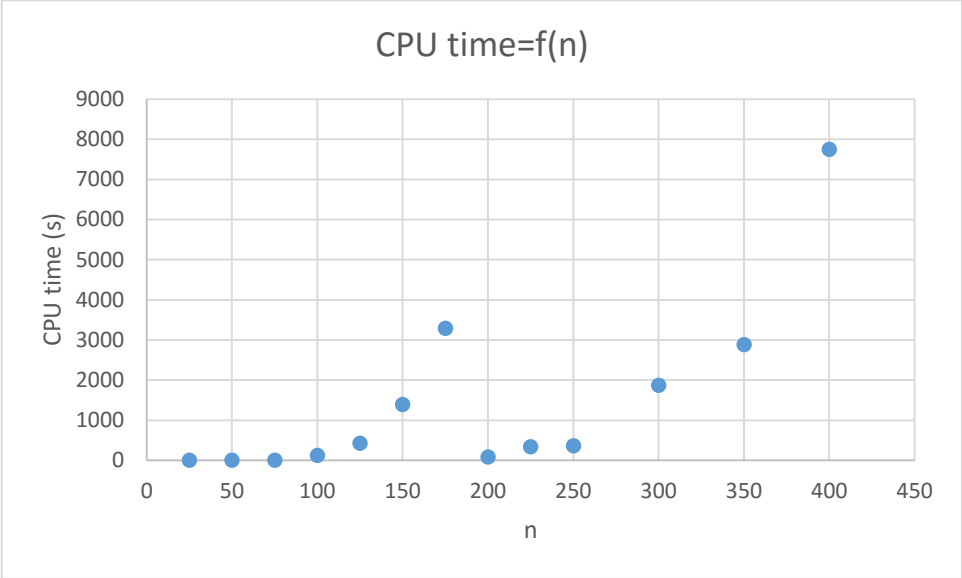
With this command, it is returned three values of time: real time, user time and system time. The real time refers to elapsed "wall clock time", like using a stop watch. The user time plus the system time is the total CPU time, may be more or less than the real time. The data obtained are in section A.4.6..

The line to n=275 is empty, because the computer was running more than 10 hours, and it could not solve it.

It is represented the real time and the CPU time as a function of the size of the chessboard (n):



Graphical representation of the real time as a function of the chessboard size (n)



Graphical representation of the CPU time as a function of the chessboard size (n)

As it could be seen, more less both graphics are the same. When the size of the chessboard grows, the time grows in more less as an exponential function. This is due to the complexity class of this problem is NP hard. With high sizes, it is impossible to solve it.

With the case of n=275, it was not found a solution in a time of more than 10 hours. Probably, it is because that size is more difficult to solve than others, so the graphics do not follow a strict exponential function, but most of the values yes.

It is tried to solve a problem with n=450, but the program was running more than 16 hours, and it was not found a solution. Probably, because the exponential function is tending to the infinity.

8. Conclusion

This research, principal focus on SoPlex, it is exposed one of the ways to solve Linear Programming problems: with LP or MPS format. As it could be seen along the work, the LP format has the advantage that is more intuitive and simple, but the problem is that SoPlex cannot read a file with row of more than 8910 characters. If it is wanted to solve a problem with a large number of variables, it is a disadvantage and it is needed to use the MPS format, more difficult and non-intuitive.

In the complexity study of both solvers, SCIP and SoPlex, it is seen very clearly, the different class problems depending on their complexity and in what is focus every solver.

SoPlex only can solve Linear Programming problems, so the execution time and memory used has a polynomial complexity tendency, function of the number of variables and other parameters.

However, SCIP is focus on mixed integer problems and it can solve NP problems. When the size of the problems grow, the complexity grows in an exponential way, as was shown in the example of the Queens Problem.

To sum up, SCIP Optimization Suite is an useful tool to solve Operation Research problems. If it is wanted to solve a LP problem, SoPlex will be the correct solver and the execution time and memory used will tend to a polynomial complexity, having always a solution. However, SCIP will be used to NP-hard problems, so only it will be found a solution if the size of the problem is not too big, due to the exponential complexity tendency.

Bibliography

[1] Hillier, F. S., Lieberman, G. J., Introduction to Operation Research: MacGraw-Hill (7th edition).

[2] Taha, H.A., Investigación de Operaciones: Pearson (7ª edition)

[3] Winston, W.L., Investigación de Operaciones, Aplicaciones y algoritmos: Thompson (4th edition).

[4] Kulej, M., Operation Research: Wrocław University of Technology, 2011.

[5] Chvátal, V., Linear Programming: W.H. Freeman & Co., 1983.

[6] Nahmias, S., Análisis de la producción y las operaciones: McGraw-Hill Ineramericana México.

[7] Rosenfeld, R., Irazábal, J., Computabilidad, complejidad computacional y verificación de programas: Edulp, 2013.

[8] http://scip.zib.de/

[9] http://soplex.zib.de/

[10] http://lpsolve.sourceforge.net

[11] Schwarz, C., An introduction to SCIP: University of Bayreuth, 2010.

Appendix

A1. MPS and LP files

A.1.1. LP file from example 1 (Section 6.3.3.1)

```
Maximize
60 x1 + 30 x2 + 20 x3

Subject to
res1: 8 x1 + 6 x2 + 1 x3 <= 48
res2: 4 x1 + 2 x2 + 1.5 x3 <= 20
res3: 2 x1 + 1.5 x2 + 0.5 x3 <= 8

Bounds
x1 >= 3
5 >= x2 >=1

End
```

A.1.2. MPS file from example 1 (Section 6.3.3.1)

```
NAME          EXAMPLE1
OBJSENSE
 MAX
ROWS
 N  OBJ
 L  RES1
 L  RES2
 L  RES3
COLUMNS
    X1        OBJ       60             RES1      8
    X1        RES2      4              RES3      2
    X2        OBJ       30             RES1      6
    X2        RES2      2              RES3      1.5
    X3        OBJ       20             RES1      1
    X3        RES2      1.5            RES3      0.5
RHS
    RHS1      RES1      48             RES2      20
    RHS1      RES3      8
BOUNDS
 LO BND1      X1        3
 LO BND1      X2        1
 UP BND1      X2        5
ENDATA
```

A.1.3. LP file from example 2 (Section 6.3.3.2.)

```
Minimize
8 x[1][1] + 6 x[1][2] + 10 x[1][3] + 9 x[1][4] + 9 x[2][1] + 12
x[2][2] + 13 x[2][3] + 7 x[2][4] + 14 x[3][1] + 9 x[3][2] + 16
x[3][3] + 5 x[3][4]

Subject to
x[1][1] + x[1][2] + x[1][3] + x[1][4] <= 35
x[2][1] + x[2][2] + x[2][3] + x[2][4] <= 50
x[3][1] + x[3][2] + x[3][3] + x[3][4] <= 40

x[1][1] + x[2][1] + x[3][1] >= 45
x[1][2] + x[2][2] + x[3][2] >= 20
x[1][3] + x[2][3] + x[3][3] >= 30
x[1][4] + x[2][4] + x[3][4] >= 30
Bounds

End
```

A.1.4. LP file from Product-Mix example (Section 6.4.2.1)

```
Maximize
15 x1 + 5 x2 + 12 x3 + 10 x4

Subject to
Res1: 1.0 x1 + 1.5 x2 + 2.0 x3 + 1.0 x4 <= 450
Res2: 3.0 x1 + 2.0 x2 + 0.0 x3 + 2.0 x4 <= 400
Res3: 2.0 x1 + 0.5 x2 + 3.0 x3 + 1.0 x4 <= 620

Bounds
200 >= x1 >= 2
200 >= x2 >= 20
150 >= x3 >= 30
100 >= x4 >= 30

End
```

A.1.5. LP file generated from Product-Mix program with random data

```
Maximize
14x1 +24x2 +28x3 +7x4

Subject to
Res1: 0x1 +5x2 +5x3 +8x4 <= 1326
Res2: 8x1 +4x2 +1x3 +8x4 <= 1248
Res3: 5x1 +5x2 +7x3 +6x4 <= 1579

Bounds
247 >= x1 >= 0
279 >= x2 >= 0
276 >= x3 >= 0
260 >= x4 >= 0

End
```

A.1.6. LP file from Transport Problem program with random data

```
Minimize
15 x1_1 + 4 x1_2 + 17 x1_3 + 8 x1_4 + 3 x2_1 + 6 x2_2 + 19 x2_3 +
3 x2_4 + 18 x3_1 + 3 x3_2 + 6 x3_3 + 3 x3_4

Subject to
x1_1 + x1_2 + x1_3 + x1_4 <= 72
x2_1 + x2_2 + x2_3 + x2_4 <= 138
x3_1 + x3_2 + x3_3 + x3_4 <= 112

x1_1 + x2_1 + x3_1 >= 40
x1_2 + x2_2 + x3_2 >= 54
x1_3 + x2_3 + x3_3 >= 113
x1_4 + x2_4 + x3_4 >= 77
Bounds

End
```

A.2. SoPlex displays on screen

A.2.1. SoPlex result of the example 1 (Section 6.3.3.1.)

```
LP has 3 rows 3 columns and 9 nonzeros.

Simplifier removed 0 rows, 0 columns, 0 nonzeros, 0 col bounds, 0 row
bounds
Reduced LP has 3 rows 3 columns 9 nonzeros
Equilibrium scaling LP
type |   time |   iters | facts |  shift   |violation |    value
  L  |    0.0 |       0 |     1 | 1.75e+01 | 1.90e+01 | 1.49999425e+02
  E  |    0.0 |       1 |     2 | 0.00e+00 | 7.50e+00 | 2.20000000e+02
  E  |    0.0 |       2 |     3 | 0.00e+00 | 0.00e+00 | 2.30000000e+02
 --- transforming basis into original space
  L  |    0.0 |       0 |     1 | 0.00e+00 | 0.00e+00 | 2.30000000e+02
  L  |    0.0 |       0 |     1 | 0.00e+00 | 0.00e+00 | 2.30000000e+02

SoPlex status       : problem is solved [optimal]
Solving time (sec)  : 0.00
Iterations          : 2
Objective value     : 2.30000000e+02


Primal solution (name, value):
x1            3.000000000
x2            1.000000000
x3            1.000000000
All other variables are zero (within 1.0e-16).
```

A.2.2. SoPlex result of the example 2 (Section 6.3.3.2.)

```
LP has 7 rows 12 columns and 24 nonzeros.

Simplifier removed 0 rows, 0 columns, 0 nonzeros, 0 col bounds, 0 row bounds
Reduced LP has 7 rows 12 columns 24 nonzeros
Equilibrium scaling LP
type |   time |   iters | facts |  shift   |violation |    value
  L  |    0.0 |       0 |     1 | 0.00e+00 | 1.25e+02 | 0.00000000e+00
  L  |    0.0 |       7 |     2 | 0.00e+00 | 0.00e+00 | 1.02000000e+03
 --- transforming basis into original space
  L  |    0.0 |       0 |     1 | 0.00e+00 | 0.00e+00 | 1.02000000e+03
  L  |    0.0 |       0 |     1 | 0.00e+00 | 0.00e+00 | 1.02000000e+03

SoPlex status       : problem is solved [optimal]
Solving time (sec)  : 0.00
Iterations          : 7
Objective value     : 1.02000000e+03


Primal solution (name, value):
x[1][2]      10.000000000
x[1][3]      25.000000000
x[2][1]      45.000000000
x[2][3]       5.000000000
x[3][2]      10.000000000
x[3][4]      30.000000000
All other variables are zero (within 1.0e-16).
```

A.2.3. SoPlex Result of the Product-Mix example

```
LP has 3 rows 4 columns and 11 nonzeros.

Simplifier removed 0 rows, 0 columns, 0 nonzeros, 0 col bounds, 0 row
bounds
Reduced LP has 3 rows 4 columns 11 nonzeros
Equilibrium scaling LP
type |   time |   iters | facts |  shift   |violation |    value
  L  |    0.0 |       0 |     1 | 0.00e+00 | 1.68e+03 | 6.80000000e+03
  L  |    0.0 |       3 |     2 | 0.00e+00 | 0.00e+00 | 3.50800000e+03
 --- transforming basis into original space
  L  |    0.0 |       0 |     1 | 0.00e+00 | 0.00e+00 | 3.50800000e+03
  L  |    0.0 |       0 |     1 | 0.00e+00 | 0.00e+00 | 3.50800000e+03

SoPlex status       : problem is solved [optimal]
Solving time (sec)  : 0.00
Iterations          : 3
Objective value     : 3.50800000e+03


Primal solution (name, value):
x1           56.000000000
x2           20.000000000
x3          134.000000000
x4           96.000000000
All other variables are zero (within 1.0e-16).
```

A.2.4. SoPlex Result of the LP file in section A.1.5.

```
LP has 3 rows 4 columns and 11 nonzeros.

Simplifier removed 0 rows, 0 columns, 0 nonzeros, 0 col bounds, 0 row
bounds
Reduced LP has 3 rows 4 columns 11 nonzeros
Equilibrium scaling LP
type |   time |   iters | facts |  shift   |violation |    value
  L  |    0.0 |       0 |     1 | 0.00e+00 | 1.23e+04 | 1.97020000e+04
  L  |    0.0 |       3 |     2 | 0.00e+00 | 0.00e+00 | 7.01704516e+03
 --- transforming basis into original space
  L  |    0.0 |       0 |     1 | 0.00e+00 | 0.00e+00 | 7.01704516e+03
  L  |    0.0 |       0 |     1 | 0.00e+00 | 0.00e+00 | 7.01704516e+03

SoPlex status       : problem is solved [optimal]
Solving time (sec)  : 0.00
Iterations          : 3
Objective value     : 7.01704516e+03


Primal solution (name, value):
x1           36.561290323
x2          230.103225806
x3           35.096774194
All other variables are zero (within 1.0e-16).
```

A.2.5. SoPlex Result of the LP file in section A.1.6.

```
LP has 7 rows 12 columns and 24 nonzeros.

Simplifier removed 0 rows, 0 columns, 0 nonzeros, 0 col bounds, 0 row
bounds
Reduced LP has 7 rows 12 columns 24 nonzeros
Equilibrium scaling LP
type |   time |   iters | facts |  shift   |violation |    value
  L  |    0.0 |       0 |     1 | 0.00e+00 | 2.84e+02 | 0.00000000e+00
  L  |    0.0 |       6 |     2 | 0.00e+00 | 0.00e+00 | 1.25600000e+03
 --- transforming basis into original space
  L  |    0.0 |       0 |     1 | 0.00e+00 | 0.00e+00 | 1.25600000e+03
  L  |    0.0 |       0 |     1 | 0.00e+00 | 0.00e+00 | 1.25600000e+03

SoPlex status       : problem is solved [optimal]
Solving time (sec)  : 0.00
Iterations          : 6
Objective value     : 1.25600000e+03


Primal solution (name, value):
x1_2          54.000000000
x1_3           1.000000000
x2_1          40.000000000
x2_4          77.000000000
x3_3         112.000000000
All other variables are zero (within 1.0e-16).
```

A.3. C programs

A.3.1. Product-Mix program of a known problem

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
	int i, j;
	int m = 3;
	int n = 4;

	float a[3][4] = { {1,1.5,2,1},{3,2,0,2},{2,0.5,3,1} };

	float b[3] = { 450,400,620 };

	float p[4] = { 25,20,30,15 };

	float c[4] = { 10,15,18,5 };

	int l[4] = { 2,20,30,30 };

	int u[4] = { 200,200,150,100 };

	float v[4];

	for(j=0;j<n;j++)
	{
		v[j] = p[j] - c[j];
	}


	freopen("ProductMixResult.lp","w",stdout);

	/*Objective Function*/

	printf("Maximize \n");

	for(j=0;j<n;j++)
	{
		printf("%.f x%d ", v[j], j+1);

		if(j<n-1)
		{
			printf("+ ");
		}
	}
		printf("\n\n");

	/*Subject to*/
	printf("Subject to \n");

	for(i=0;i<m;i++)
	{
		printf("Res%d: ", i+1);

		for(j=0;j<n;j++)
		{
			printf("%.1f x%d ", a[i][j], j+1);

			if (j<n-1)
```

```
                {
                        printf("+ ");
                }
        }
        printf("<= %.f \n", b[i]);
    }

    printf("\n");

    /*Bounds*/
    printf("Bounds \n");

    for(j=0;j<n;j++)
    {
        printf("%d >= x%d >= %d \n", u[j], j+1, l[j]);
    }

    printf("\n");
    printf("End");

    fclose(stdout);

}
```

A.3.2. Product-Mix program with random data

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i, j, m, n;
    int u_deviation, l_deviation;
    float sum;
    float seed;

    FILE *f1;
    int vector[4];
    f1 = fopen("ProductMix_Data.txt", "r");

    if((f1!=NULL))
    {
        for (i = 0;i<4;i++)
        {
            fscanf(f1, "%d", &vector[i]);
        }
    }

    else
    {
        printf("Error opening data file");
    }
    printf("\n");

    n = vector[0];
    m = vector[1];
    u_deviation = vector[2];
    l_deviation = vector[3];



    float **a;
```

```
a = (float **)malloc(m * sizeof(float *));

for (i = 0; i<m; i++)
{
	a[i] = (float *)malloc(n * sizeof(float));
}

float *b;
b = (float *)malloc(m * sizeof(float));

float *p;
p = (float *)malloc(n * sizeof(float));

float *c;
c = (float *)malloc(n * sizeof(float));

float *v;
v = (float *)malloc(n * sizeof(float));

int *l;
l = (int *)malloc(n * sizeof(int));

int *u;
u = (int *)malloc(n * sizeof(int));

float *x;
x = (float *)malloc(n * sizeof(float));


seed = 4853;
srand(seed);

/*Matrix of product time in each resource*/
for(i=0;i<m;i++)
{
	for (j = 0;j<n;j++)
	{
		a[i][j] = rand() % 11;
	}
}

/*Random solution*/
for (j = 0; j < n; j++)
{
	x[j] = rand() % 101;
}

/*Capacity vector*/
for (i = 0; i < m; i++)
{
	sum = 0;
	for (j = 0; j < n; j++)
	{
		sum = sum + a[i][j] * x[j];
	}

	b[i] = sum + rand() % 101;
}

/*Minimun bound of each product*/
	for(j=0;j<n;j++)
{
```

```
        l[j] = x[j] - l_deviation;

        if(l[j]<0)
        {
                l[j] = 0;
        }
}

/*Maximun bound of each product*/
for (j = 0; j<n; j++)
{
        u[j] = x[j] + u_deviation;
}

/*Prize of each product*/
for(j=0;j<n;j++)
{
        p[j] = 20 + rand() % 21;
}

/*Cost of each product*/
for(j=0;j<n;j++)
{
        c[j] = 10 + rand() % 11;
}


/*Profit margin of each product*/
for(j=0;j<n;j++)
{
        v[j] = p[j] - c[j];
}



/* Generate LP file*/

freopen("ProductMixResult.lp","w",stdout);

printf("Maximize \n");

/*Objective Function*/


for(j=0;j<n;j++)
{
        printf("%.fx%d ", v[j], j+1);

        if(j<(n-1))
        {
                printf("+");
        }
}

printf("\n\n");

/*Subject to*/

printf("Subject to \n");

for(i=0;i<m;i++)
```

```
        {
                printf("Res%d: ", i+1);

                for(j=0;j<n;j++)
                {
                        printf("%.fx%d ", a[i][j], j+1);

                        if (j<(n-1))
                        {
                                printf("+");
                        }
                }

                printf("<= %.f \n", b[i]);
        }

        printf("\n");

        /*Bounds*/

        printf("Bounds \n");

        for(j=0;j<n;j++)
        {
                printf("%d >= x%d >= %d \n", u[j], j+1, l[j]);
        }

        printf("\n");
        printf("End");

        fclose(stdout);


}
```

A.3.3. Transport program of a known problem

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
        int i, j;
        int m = 3;
        int n = 4;

        int c[3][4] = { {8,6,10,9},{9,12,13,7},{14,9,16,5} };

        int a[3] = { 35,50,40 };

        int b[4] = { 45,20,30,30 };



        /* Generate LP file*/

        freopen("TraProResult.lp","w",stdout);

        printf("Minimize \n");
```

```
/*Objective Function*/


for(i=0;i<m;i++)
{
	for (j = 0;j<n;j++)
	{
		printf("%dx%d_%d ", c[i][j], i + 1, j + 1);
		if(j<n-1)
		{
			printf("+ ");
		}
	}
	if(i<m-1)
	{
		printf("+ ");
	}
}

printf("\n\n");

/*Subject to*/

printf("Subject to \n");

for(i=0;i<m;i++)
{
	for (j = 0;j<n;j++)
	{
		printf("x%d_%d ", i + 1, j + 1);
		if (j<n - 1)
		{
			printf("+ ");
		}
	}

	printf("<= %d \n", a[i]);
}

printf("\n");

for(j=0;j<n;j++)
{
	for (i = 0;i<m;i++)
	{
		printf("x%d_%d ", i + 1, j + 1);
		if (i<m - 1)
		{
			printf("+ ");
		}
	}

	printf(">= %d \n", b[j]);
}

/*Bounds*/

printf("Bounds \n");
```

```
        printf("\n");
        printf("End");

        fclose(stdout);


}
```

A.3.4. Transport program with random data

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
        int i, j, m, n;
        int seed, sum;
        int demand_deviation, capacity_deviation;


        FILE *f1;
        int vector[4];
        f1 = fopen("TraPro_Data.txt", "r");

        if ((f1 != NULL))
        {
                for (i = 0; i<4; i++)
                {
                        fscanf(f1, "%d", &vector[i]);
                }
        }

        else
        {
                printf("Error opening data file");
        }
        printf("\n");

        n = vector[0];
        m = vector[1];
        demand_deviation = vector[2];
        capacity_deviation = vector[3];


        int **c;
        c = (int **)malloc(m * sizeof(int *));

        for (i = 0; i<m; i++)
        {
                c[i] = (int *)malloc(n * sizeof(int));
        }

        int *a;
        a = (int *)malloc(m * sizeof(int));

        int *b;
        b = (int *)malloc(n * sizeof(int));

        int **x;
```

```
x = (int **)malloc(m * sizeof(int *));

for (i = 0; i<m; i++)
{
	x[i] = (int *)malloc(n * sizeof(int));
}


seed = 4574;
srand(seed);



/*Random solution*/
for (i = 0; i<m; i++)
{
	for (j = 0; j<n; j++)
	{
		x[i][j] = rand() % 51;
	}
}

/*Cost Matrix*/
for(i=0;i<m;i++)
{
	for (j = 0;j<n;j++)
	{
		c[i][j] = rand() % 21;
	}
}


/*Demand Vector*/
for(j=0;j<n;j++)
{
	sum = 0;
	for(i=0;i<m;i++)
	{
		sum = sum + x[i][j];
	}
	b[j] = sum - rand() % (demand_deviation + 1);
}


/*Capacity vector*/
for(i=0;i<m;i++)
{
	sum = 0;
	for(j=0;j<n;j++)
	{
		sum = sum + x[i][j];
	}
	a[i] = sum + rand() % (capacity_deviation + 1);
}


/* Generate LP file*/

freopen("TraProResult.lp","w",stdout);

printf("Minimize \n");
```

```
/*Objective Function*/


for(i=0;i<m;i++)
{
	for (j = 0;j<n;j++)
	{
		printf("%d x%d_%d ", c[i][j], i + 1, j + 1);
		if(j<n-1)
		{
			printf("+ ");
		}
	}
	if(i<m-1)
	{
		printf("+ ");
	}
}

printf("\n\n");

/*Subject to*/

printf("Subject to \n");

for(i=0;i<m;i++)
{
	for (j = 0;j<n;j++)
	{
		printf("x%d_%d ", i + 1, j + 1);
		if (j<n - 1)
		{
			printf("+ ");
		}
	}

	printf("<= %d \n", a[i]);
}

printf("\n");

for(j=0;j<n;j++)
{
	for (i = 0;i<m;i++)
	{
		printf("x%d_%d ", i + 1, j + 1);
		if (i<m - 1)
		{
			printf("+ ");
		}
	}

	printf(">= %d \n", b[j]);
}

/*Bounds*/

printf("Bounds \n");
```

```
        printf("\n");
        printf("End");

        fclose(stdout);


}
```

A.4. Tables

A.4.1. SoPlex result of Product-Mix if the deviation has influenced

Seed	Resources	Products	qmax desviation	qmin desviation	Iterations	Solving time Soplex	Memory Before (MB)	Memory After (MB)	Memory Diference (MB)
4853	500	50	500	500	87	0,02	3094	3096	2
4853	500	50	200	200	94	0,02	3133	3133	0
4853	500	50	50	50	84	0	3100	3100	0
4853	500	100	500	500	425	0,05	3076	3076	0
4853	500	100	200	200	330	0,03	3110	3112	2
4853	500	100	50	50	247	0,03	3090	3090	0
4853	500	200	500	500	731	0,12	3104	3104	0
4853	500	200	200	200	892	0,14	3093	3093	0
4853	500	200	50	50	345	0,05	3086	3086	0
4853	500	300	500	500	1009	0,17	3223	3223	0
4853	500	300	200	200	1260	0,19	3220	3220	0
4853	500	300	50	50	876	0,11	3211	3211	0
4853	500	400	500	500	1505	0,25	2881	2885	4
4853	500	400	200	200	1188	0,2	2857	2860	3
4853	500	400	50	50	1893	0,23	2887	2891	4
4853	500	500	500	500	1292	0,36	2864	2864	0
4853	500	500	200	200	1901	0,34	2852	2852	0
4853	500	500	50	50	1329	0,2	2870	2870	0
4853	500	600	500	500	1727	0,39	2859	2864	5
4853	500	600	200	200	2436	0,5	2841	2843	2
4853	500	600	50	50	1976	0,34	2911	2911	0
4853	500	700	500	500	3252	0,88	2896	2904	8
4853	500	700	200	200	2831	0,66	2892	2905	13
4853	500	700	50	50	1360	0,27	2897	2907	10
4853	500	800	500	500	3440	0,78	2937	2946	9
4853	500	800	200	200	5592	1,17	2947	2976	29
4853	500	800	50	50	2603	0,45	2937	2944	7
4853	500	900	500	500	2597	0,61	2945	2979	34
4853	500	900	200	200	3280	0,86	2949	2986	37
4853	500	900	50	50	3982	0,86	2951	2991	40
4853	500	1000	500	500	5357	1,52	2965	3003	38
4853	500	1000	200	200	3616	1,12	2959	3000	41
4853	500	1000	50	50	7030	1,58	2995	3023	28

A.4.2. SoPlex result of Product-Mix (resources and products by hundreds)

Seed	Resources	Products	qmax deviation	qmin deviation	Iterations	Solving time Soplex	Memory Before (MB)	Memory After (MB)	Memory Diference (MB)
4853	100	100	500	500	126	0	2587	2587	0
4853	100	200	500	500	170	0,02	2575	2575	0
4853	100	300	500	500	435	0,03	2598	2598	0
4853	100	400	500	500	806	0,05	2595	2597	2
4853	100	500	500	500	691	0,03	2573	2573	0
4853	100	600	500	500	1307	0,06	2590	2590	0
4853	100	700	500	500	567	0,08	2501	2501	0
4853	100	800	500	500	1498	0,12	2484	2487	3
4853	100	900	500	500	1482	0,16	2474	2476	2
4853	100	1000	500	500	1220	0,14	2477	2479	2
4853	200	100	500	500	205	0,01	2449	2449	0
4853	200	200	500	500	239	0,03	2452	2452	0
4853	200	300	500	500	980	0,08	2469	2472	3
4853	200	400	500	500	1081	0,11	2457	2468	11
4853	200	500	500	500	753	0,09	2462	2469	7
4853	200	600	500	500	1440	0,22	2473	2490	17
4853	200	700	500	500	1081	0,17	2477	2488	11
4853	200	800	500	500	1500	0,27	2470	2470	0
4853	200	900	500	500	1645	0,31	1775	1779	4
4853	200	1000	500	500	1707	0,34	1775	1794	19
4853	300	100	500	500	197	0,03	1760	1760	0
4853	300	200	500	500	552	0,05	1760	1762	2
4853	300	300	500	500	596	0,08	1757	1757	0
4853	300	400	500	500	1278	0,16	1759	1759	0
4853	300	500	500	500	1025	0,19	1743	1756	13
4853	300	600	500	500	1376	0,19	1740	1747	7
4853	300	700	500	500	1886	0,33	1742	1744	2
4853	300	800	500	500	2126	0,42	1733	1734	1
4853	300	900	500	500	2407	0,55	1735	1736	1
4853	300	1000	500	500	2356	0,58	1739	1759	20
4853	400	100	500	500	179	0,03	1744	1744	0
4853	400	200	500	500	588	0,09	1746	1746	0
4853	400	300	500	500	1043	0,14	1751	1752	1
4853	400	400	500	500	1739	0,3	1739	1742	3
4853	400	500	500	500	1582	0,31	1742	1746	4
4853	400	600	500	500	2554	0,5	1736	1756	20
4853	400	700	500	500	2130	0,44	1739	1761	22
4853	400	800	500	500	3373	0,73	1736	1764	28
4853	400	900	500	500	3182	0,78	1733	1761	28

4853	400	1000	500	500	4024	0,86	1732	1769	37
4853	500	100	500	500	425	0,05	1737	1737	0
4853	500	200	500	500	731	0,12	1736	1736	0
4853	500	300	500	500	1009	0,17	1736	1737	1
4853	500	400	500	500	1505	0,25	1735	1736	1
4853	500	500	500	500	1292	0,36	1734	1736	2
4853	500	600	500	500	1727	0,39	1733	1735	2
4853	500	700	500	500	3252	0,88	1734	1761	27
4853	500	800	500	500	3440	0,78	1732	1758	26
4853	500	900	500	500	2597	0,61	1718	1752	34
4853	500	1000	500	500	5357	1,52	1721	1757	36
4853	600	100	500	500	386	0,08	1722	1723	1
4853	600	200	500	500	763	0,14	1723	1723	0
4853	600	300	500	500	846	0,19	1722	1722	0
4853	600	400	500	500	1009	0,36	1718	1725	7
4853	600	500	500	500	1577	0,34	1718	1722	4
4853	600	600	500	500	2068	0,48	1750	1756	6
4853	600	700	500	500	3218	0,95	1761	1795	34
4853	600	800	500	500	3568	1,12	1762	1800	38
4853	600	900	500	500	3311	1,16	1762	1804	42
4853	600	1000	500	500	5064	1,8	1760	1807	47
4853	700	100	500	500	302	0,06	1753	1754	1
4853	700	200	500	500	505	0,09	1753	1756	3
4853	700	300	500	500	929	0,22	1752	1757	5
4853	700	400	500	500	1150	0,3	1751	1757	6
4853	700	500	500	500	1459	0,41	1746	1775	29
4853	700	600	500	500	2387	0,69	1746	1777	31
4853	700	700	500	500	3684	1,28	1748	1787	39
4853	700	800	500	500	4424	1,91	1749	1792	43
4853	700	900	500	500	3968	1,16	1743	1792	49
4853	700	1000	500	500	3565	1,7	1741	1796	55
4853	800	100	500	500	292	0,06	1783	1783	0
4853	800	200	500	500	472	0,11	1784	1785	1
4853	800	300	500	500	732	0,19	1784	1791	7
4853	800	400	500	500	1198	0,38	1784	1798	14
4853	800	500	500	500	1696	0,45	1787	1804	17
4853	800	600	500	500	1903	0,66	1788	1820	32
4853	800	700	500	500	3016	1,02	1784	1830	46
4853	800	800	500	500	2552	1	1782	1832	50
4853	800	900	500	500	3059	1,23	1778	1833	55
4853	800	1000	500	500	4081	1,78	1776	1840	64
4853	900	100	500	500	277	0,08	1779	1779	0
4853	900	200	500	500	1188	0,27	1778	1783	5
4853	900	300	500	500	1193	0,33	1775	1805	30

4853	900	400	500	500	1266	0,44	1784	1814	30
4853	900	500	500	500	1674	0,55	1780	1816	36
4853	900	600	500	500	3454	1,14	1769	1815	46
4853	900	700	500	500	3630	1,36	1773	1823	50
4853	900	800	500	500	2755	1,12	1765	1817	52
4853	900	900	500	500	3522	1,53	1765	1826	61
4853	900	1000	500	500	4017	1,94	1769	1838	69
4853	1000	100	500	500	414	0,09	1771	1772	1
4853	1000	200	500	500	701	0,31	1768	1773	5
4853	1000	300	500	500	1359	0,45	1768	1770	2
4853	1000	400	500	500	1719	0,59	1770	1800	30
4853	1000	500	500	500	2301	0,91	1770	1811	41
4853	1000	600	500	500	1889	1,14	1772	1819	47
4853	1000	700	500	500	3219	1,3	1778	1832	54
4853	1000	800	500	500	6033	2,34	1779	1841	62
4853	1000	900	500	500	6032	2,33	1874	1947	73
4853	1000	1000	500	500	5634	2,84	1873	1950	77

A.4.3. SoPlex result of Product-Mix (resources by thousands, n=1000)

Seed	Resources	Products	qmax deviation	qmin deviation	Iterations	Solving time Soplex	Memory Before (MB)	Memory After (MB)	Memory Diference (MB)
4853	1000	1000	500	500	5634	2,84	2835	2914	79
4853	2000	1000	500	500	5998	6,22	2890	3036	146
4853	3000	1000	500	500	8151	12,97	2827	3055	228
4853	4000	1000	500	500	6576	15,23	2749	3091	342
4853	5000	1000	500	500	10056	33,36	2745	3159	414
4853	6000	1000	500	500	8142	34	2742	3234	492
4853	7000	1000	500	500	8261	50,02	2804	3379	575
4853	8000	1000	500	500	11163	67,55	2799	3424	625
4853	9000	1000	500	500	10810	95,88	2819	3508	689
4853	10000	1000	500	500	10390	88,52	2340	3082	742
4853	11000	1000	500	500	10529	114,8	2223	3083	860
4853	12000	1000	500	500	8017	111,2	2192	3117	925
4853	13000	1000	500	500	10342	169,44	2214	3222	1008
4853	14000	1000	500	500	9977	201,12	2192	3291	1099
4853	15000	1000	500	500	10114	202,78	2198	3373	1175

A.4.4. Soplex result of Product-Mix (resources by 5 thousands, n=1000)

Seed	Resources	Products	qmax deviation	qmin deviation	Iterations	Solving time Soplex	Memory Before (MB)	Memory After (MB)	Memory Diference (MB)
4853	1000	1000	500	500	5634	2,84	2835	2914	79
4853	5000	1000	500	500	10056	33,36	2745	3159	414
4853	10000	1000	500	500	10390	88,52	2340	3082	742
4853	15000	1000	500	500	10114	202,78	2198	3373	1175
4853	20000	1000	500	500	11604	388,12	2171	3785	1614
4853	25000	1000	500	500	10773	598,09	1961	4093	2132
4853	30000	1000	500	500	11776	854,17	2105	4287	2182
4853	35000	1000	500	500	11410	1254,8	2122	4766	2644
4853	40000	1000	500	500	12892	2015,56	2227	5052	2825
4853	45000	1000	500	500	11864	2126,64	2556	5784	3228
4853	50000	1000	500	500	12454	2278,88	2693	6325	3632
4853	55000	1000	500	500	11859	3119,69	3241	7135	3894
4853	60000	1000	500	500	11698	3472,7	2854	6804	3950
4853	65000	1000	500	500	14156	4594,44	2623	7045	4422
4853	70000	1000	500	500	12723	4992,98	2454	7182	4728
4853	75000	1000	500	500	13058	5948,91	2595	7685	5090

A.4.5. Soplex result of Tranport Problem

Seed	Sources	Destinations	Demand Deviation	Capacity Deviation	Iterations	Solving time Soplex	Memory Diference (MB)	Number of variables
4574	10	10	10	20	21	0	0	100
4574	10	50	10	20	60	0	0	500
4574	10	75	10	20	97	0	0	750
4574	10	80	10	20	108	0	0	800
4574	10	85	10	20	114	0	0	850
4574	10	86	10	20	114	0	0	860
4574	10	87	10	20	Error (more than 8190 characters)			870
4574	10	90	10	20	Error (more than 8190 characters)			900
4574	20	10	10	20	38	0	0	200
4574	20	20	10	20	55	0	0	400
4574	20	30	10	20	68	0	0	600
4574	20	40	10	20	81	0	0	800
4574	20	50	10	20	Error (more than 8190 characters)			1000
4574	30	10	10	20	50	0	0	300
4574	30	20	10	20	64	0	0	600
4574	30	30	10	20	Error (more than 8190 characters)			900

A.4.6. SCIP result

n	Real Time			User Time			System time			CPU time
	min	s	Total (s)	min	s	Total (s)	min	s	Total (s)	Total (s)
25		1,504	**1,504**		0,246	0,246		0,275	0,275	**0,521**
50		1,487	**1,487**		1,123	1,123		0,381	0,381	**1,504**
75		3,119	**3,119**		2,918	2,918		0,213	0,213	**3,131**
100	2	43,175	**163,175**	2	0,694	120,694		0,694	0,694	**121,388**
125	7	7,211	**427,211**	7	1,998	421,998		1,322	1,322	**423,32**
150	23	12,532	**1392,53**	23	2,746	1382,75		3,257	3,257	**1386,003**
175	55	29,116	**3329,12**	54	43,623	3283,62		6,865	6,865	**3290,488**
200	1	18,646	**78,646**	1	17,73	77,73		0,727	0,727	**78,457**
225	5	38,815	**338,815**	5	37,262	337,262		1,025	1,025	**338,287**
250	6	32,592	**392,592**	6	0,592	360,592		1,385	1,385	**361,977**
275										
300	31	14,512	**1874,51**	31	0,808	1860,81		3,008	3,008	**1863,816**
350	68	17,556	**4097,56**	47	56,074	2876,07		4,153	4,153	**2880,227**
400	188	37,002	**11317**	128	56,046	7736,05		9,755	9,755	**7745,801**

A.5. SCIP displays on screen

A.5.1. Queens problem test

```
for i in 1 2 4 8 16 ; do \
                echo ; echo "Testing the n-queens solver with n = $i" ; \
                bin/queens.cygwin.x86_64.gnu.opt.spx $i || break 1 ; \
                echo ; \
        done
```

```
Testing the n-queens solver with n = 1
*****************************************
* n-queens solver based on SCIP             *
*                                           *
* (c) Cornelius Schwarz (2007)              *
*****************************************

solution for 1-queens:

 ---
| D |
 ---


Testing the n-queens solver with n = 2
*****************************************
* n-queens solver based on SCIP             *
*                                           *
* (c) Cornelius Schwarz (2007)              *
*****************************************

solution for 2-queens:

no solution found


Testing the n-queens solver with n = 4
*****************************************
* n-queens solver based on SCIP             *
*                                           *
* (c) Cornelius Schwarz (2007)              *
*****************************************

solution for 4-queens:

 --- --- --- ---
|   |   | D |   |
 --- --- --- ---
| D |   |   |   |
 --- --- --- ---
|   |   |   | D |
 --- --- --- ---
|   | D |   |   |
 --- --- --- ---


Testing the n-queens solver with n = 8
*****************************************
* n-queens solver based on SCIP             *
*                                           *
* (c) Cornelius Schwarz (2007)              *
*****************************************

solution for 8-queens:

 --- --- --- --- --- --- --- ---
|   | D |   |   |   |   |   |   |
 --- --- --- --- --- --- --- ---
|   |   |   | D |   |   |   |   |
 --- --- --- --- --- --- --- ---
|   |   |   |   |   | D |   |   |
 --- --- --- --- --- --- --- ---
|   |   |   |   |   |   |   | D |
 --- --- --- --- --- --- --- ---
|   |   | D |   |   |   |   |   |
```

```
 --- --- --- --- --- --- --- ---
| D |   |   |   |   |   |   |   |
 --- --- --- --- --- --- --- ---
|   |   |   |   |   |   | D |   |
 --- --- --- --- --- --- --- ---
|   |   |   |   | D |   |   |   |
 --- --- --- --- --- --- --- ---
```

```
Testing the n-queens solver with n = 16
*****************************************
* n-queens solver based on SCIP         *
*                                       *
* (c) Cornelius Schwarz (2007)          *
*****************************************

solution for 16-queens:
```