A Recommender Framework for Portfolio Selection to Reduce Fraud in Electricity Distribution Using Integer Programming and Heuristic Techniques

Luchi 1, Fábio Fabris 2, Marcos Daniel Valadão Baroni, Alexandre Rodrigues, Flávio Miguel Varejão

Núcleo de Inferência em Algoritmos (NINFA) - Departamento de Informática
Universidade Federal do Espírito Santo
Vitória, Espírito Santo, Brasil

Abstract. Electricity fraud corresponds to a major source of loss in Electricity Distribution Companies in developing countries. To attenuate this, companies have at their disposal several loss reduction actions. These actions must be performed to reach a given electricity loss reduction goal, established by the regulatory agency. If this goal is not reached, the profit margin of the company is reduced. This work defines a formal model for the action allocation task and proposes an exact mathematic programming method for solving small instances of the problem and two heuristic methods for larger ones. We compare the techniques considering solution quality.

1 Introduction

Electricity loss due to fraud is a major concern among Electricity Distribution Companies (EDCO's) in developing countries. The current estimate in our local Brazilian distributer is that about 13% of the overall distributed electricity is lost due to fraud (non-technical losses) whereas only 7% is lost due to inherent physical phenomena involved in the transmission of electricity (technical losses) [**ANEEL2012**]. The excessive non-technical loss represents a profit reduction for the EDCOS's and ultimately an impact on the quality of the distribution service, which in turn, motivate even more non-technical losses.

In principle, EDCO's could increase the charge of consumers to payoff the loss of electricity due to non-technical sources. However, the Brazilian regulatory agency of electricity (ANEEL) dictates that only a given amount of non-technical loss may be covered by increasing the charges of non-fraudulent clients. This electricity threshold is known as *non-technical loss reducing goal.* The electricity loss above the goal cannot be paid by consumers and results in profit reduction to the EDCO's.

In practice, the existence of a non-technical loss reducing goal means that EDCO's are forbidden to charge clients more than a fixed value per Kilowatt/hour to attenuate losses due to non-technical sources. The non-technical loss reducing goal (and indirectly the maximum Kilowatt/hour charge) is established by ANEEL by studying the EDCO's profile and the historical non-technical losses behaviour of similar regions of the country. Additionally, the EDCO's are required to *reduce* the Kilowatt/hour electricity charge if they manage to mitigate electricity loss more than the established non-technical loss reducing goal. This means that there is no gain to the EDCO's if they reduce the non-technical losses more than the established goal.

Usually, the non-technical loss reducing goal is given for the following three years, defining a *non-technical loss reducing curve*, so that EDCO's may plan their *non-technical loss reducing actions* for the following years. If EDCO's don't meet the imposed non-technical loss reduction curve in a given year, their profit in that year is reduced by the corresponding value of the electricity under the goal. For instance, if the regulatory agency establishes that the non-technical loss reduction goal is $40\,GW$, $50\,GW$, $60\,GW$, and the EDCO achieves $30\,GW$, $40\,GW$, $60\,GW$; $20\,GW$ won't be sold to consumers and will ultimately represent a profit loss. However, if the EDCO's reduces more electricity loss than the established goal in a year, they are required to reduce their gains to match the profit that they would have had in that year if they were exactly on the non-technical loss reduction goal.

This scenario introduces a new problem to the EDCO's: given an investment portfolio comprised of several non-technical loss reducing actions, a non-technical loss reduction curve and a yearly budget, which is the best possible allocation of actions that maximizes the return of the investment? That is, the allocation with the greatest profit or, in other words, the best *investment strategy.* For instance, if the EDCO chooses to do nothing, it would not be able to charge for some portion of the distributed electricity, since this usually costs more than performing the non-technical loss reducing actions, doing nothing would not be the optimal course of action. On the other hand, reduce the loss to a greater extent than the imposed reduction curve is also not the optimal decision, since the over-reduced electricity loss does not increase the overall EDCO profit.

The current methodology used by EDCO's is to manually select the fraud reducing actions in a heuristic trial-and-error fashion. As we shall present in this work, choosing the best non-technical loss reducing actions is a complex combinatorial problem, thus the current *de facto* procedure hardly finds to optimum solution of the problem. Since reducing technical losses is an important activity of EDCOS's, this motivates the use of computational techniques to find the best, or at least good, solutions to our problem.

1.1 Organization

The reminder of the paper is organized as follows: Section 2 defines the problem of choosing non-technical loss reducing actions more formally. Section 3 is comprised of an investigation of related works in the literature and a explanation of the inherent complexity of our problem. Section 4 introduces our methods for solving the defined problem. In section 5 we present our experimental evaluation.

Finally, in section 6 we make our concluding remarks about the experimental results and lay ground for future work.

2 Problem Definition

To apply optimization algorithms in our problem we first need to define it formally. Lets assume that we must maximize the *net present value* (the present investment return considering inflation) for a reduction plan of M years, given:

- a yearly budget for a set of L resources o_{il}, $1 \leq i \leq M$, $1 \leq l \leq L$;
- a fraud reduction goal g_i, $1 \leq i \leq M$ that represents the amount of electricity loss that must be reduced;
- an *internal rate of return* r, that represents the yearly depreciation of the investment (constant for all investments and years).

We also assume that there are several actions to choose from a portfolio of actions of size N. Each action j, $1 \leq j \leq N$, contained in the portfolio has several parameters:

- the electricity value v_j, that represents the value of the unit of electricity for each action j in portfolio, i.e., the value of each Kilowatt of electricity;
- m_j, the maximum number of times that action j may be performed, we shall refer to it as *the market* of the action;
- $u_{j,i}$ the maximum number of times that action j may be performed in the i-th year;
- $c_{j,l}$, the l-th cost associated to each execution of action j;
- $e_{j,k}$, the amount of lost electricity that the action j will reduce after k years it was executed, when taken once;
- a set $D_j \subset \mathbb{N}^*$ that represents which actions must be performed prior action j. For each action j the action $D_{j,d}$ $(1 \leq d \leq |D_j|)$ must be executed a number of times defined by
- the parameter $Q_{j,d} \in \mathbb{R}^+$.

Our objective is to find a set of values for variables $x_{j,i}$, $\forall i, j$, $x_{j,i} \in \mathbb{N}$, the number of times that the action j will be performed in the i-th year. We wish to choose a combination of values that maximizes the net present value and is under the fraud reduction goal for all years.

The constraints of the problem are the yearly budget restriction,

$$\sum_{j=1}^{N} x_{j,i} \cdot c_{j,l} \leq o_{i,l} \, \forall i, l, \tag{1}$$

the market restriction,

$$\sum_{i=1}^{M} x_{j,i} \leq m_j \, \forall j, \tag{2}$$

the maximum number of times the action $x_{j,i}$ may be performed in i-th year,

$$x_{j,i} \leq u_{j,i}\, \forall j, i, \tag{3}$$

the goal restriction,

$$\sum_{j=1}^{N} \sum_{k=1}^{M} R_{i,j,k}(\bar{x}) \leq g_i\, \forall i, \tag{4}$$

$R_{i,j,k}$ being the fraud reduction in the k-th year by the action j taken on i-th year, defined as

$$R_{i,j,k}(\bar{x}) = x_{j,i} \cdot e_{j,k-i+1}\, \forall k \geq i, \tag{5}$$

and the dependency restriction for all actions and for all years,

$$\forall j, k \quad \sum_{i=1}^{k} x_{d,i} \geq x_{j,d} \cdot Q_{j,d}\, \forall d \in D_j. \tag{6}$$

To introduce the objective function we must define some auxiliary concepts: The yearly total cost, C_i,

$$C_i(\bar{x}) = \sum_{j=1}^{N} \sum_{l=1}^{L} x_{j,i} \cdot c_{j,l}\, \forall i, \tag{7}$$

the gained profit in i-th year, due to elimination of the fraud, V_i,

$$V_i(\bar{x}) = \sum_{j=1}^{N} \sum_{k=1}^{M} R_{k,j,i}(\bar{x}) \cdot v_j\, \forall i, \tag{8}$$

by definition $V_i - C_i$ is the total cash flow in i-th year.

Finally, we may define the objective function as:

$$max(O(\bar{x})) = max \left(\sum_{i=1}^{M} \frac{V_i(\bar{x}) - C_i(\bar{x})}{(1+r)^i} \right). \tag{9}$$

The objective function represents the net present value, that is, the sum of the yearly cash flows, $V_i - C_i$, adjusted by the internal rate of return for all years.

2.1 Realistic Version

To adapt the previously defined problem to the current state of our local EDCO, we simplify it in several ways:

- We consider that the yearly budgets are insufficient to surpass the goal for any given year;
- We consider all actions on portfolio have an positive cash flow;
- We ignore the dependencies among actions, that is, the set D_j is empty for all actions;

- We assume that both the vectors that define the costs of the j-th action c_j and the i-th budget o_i have size 1;
- We assume that the electricity value v_j is constant for all actions;
- We assume that the value of $\sum_{i=0}^{N} u_{j,i} \leq m_j, \forall j$.

The simplifications may be altered accordingly to the current state of the distributer. Other companies may eliminate or consider other restrictions, and as far as we know, the model is general enough to model for the majority of possible scenarios.

3 Related Work

The exact formulation of the problem has shown to be quite particular and no work addressing a similar problem was found in literature. If the depreciation of the investments is not considered ($r = 0$), the problem may be viewed as a *Partially-Ordered Multidimensional Multiple Knapsack Problem* (POMMKP). Which is, as far as we know, a novel version of the classical Knapsack Problem.

Additionally, If we consider just the cost $c_{j,l}$ of each action (weight), the reduction of lost electricity $e_{j,k}$ of each action (profit) and the yearly budget (capacity) on a single year situation ($M = 1$) the problem could be classified as a simple *Knapsack Problem* [**pisinger1995**]. If we now add the dependency restriction, the problem could be considered a *Partially-Ordered Knapsack Problem* [**pok2002**]. If we consider multiple years ($M > 1$) instead of the dependency restriction, it would characterize a *Multiple Knapsack Problem*, with each year with its own capacity. Considering multiple resources ($M = 1, L > 1$) and no dependency, the problem can be formulated as a *Multidimensional Knapsack Problem.*

All the problems mentioned above are hard to solve optimally, these problems are known as $\mathcal{NP}$-hard problems and are no know polynomial time algorithms to solve, unless $\mathcal{P} = \mathcal{NP}$. For the classical knapsack problem there is a FPTAS (*Fully Polynomial Time Approximation Scheme*), while for the others variants mentioned above are hard to aproximate and there are only PTAS (*Polynomial Time Approximation Scheme*) to solve them with a certain degree of approximation of the optimal solution, [**pok2002**, **puchinger2006core**, **dawande2000approximation**].

The POMMKP can be considered as a generalization of the above mentioned problems, thus we may conclude that POMMKP is at least as difficult as any of them. Since our problem is a generalization of the POMMKP, it is as difficult as the POMMKP. For this reason there is no algorithm that solves our problem in polynomial time (considering $\mathcal{P} \neq \mathcal{NP}$). Given the difficulty of approximating the optimal solution of this problem through exact algorithms, heuristics methods are necessary for solving larger instances of the problem.

4 Solving the Problem

To solve our allocation problem we have developed three methodologies: 1) an exact algorithm using integer programming, 2) a heuristic Gradient Ascent algorithm using the LP-relaxed version of the integer programming model as a starting solution (GALP) and 3) a Tabu Search algorithm that also uses the LP solution as a starting point (TSLP). In the following subsections we present each approach and comment on their particularities.

4.1 Exact Mathematical Programming Approach

The exact approach considers the realistic version of the MIP formulation, previously defined in Section 2. To solve this formulation, we have used a branch-and-bound method [**lawler1966branch**] included in the GLPK solver [**GLPK**].

4.2 Gradient Ascent using the LP solution (GALP)

We use a simple heuristic approach, namely Gradient Ascent (GA), using the truncated solution of the relaxed version of the linear problem (considering $x_{j,i} \in [0, u_{j,i}]$) as a initial search point. Since the relaxed version of the linear problem is easily solvable by the simplex algorithm [**dantzig1955generalized**] and the Gradient Ascent algorithm has fast convergence characteristics, this approach is very efficient, converging in less than 10 iterations in most of our test instances. However because it is a greedy heuristic, it has the flaw of getting stuck in local minima, because it never accepts a solution if it is worse than the current one. Algorithm 1 depicts the procedure.

We have compared the solution quality randomly starting the *GA* algorithm in different locations of the search space and found that in our experiments it never found a better solution than using the truncated Linear Problem solution as a start.

Algorithm 1 Gradient Ascent Algorithm

```
 1: function GRADIENT ASCENT(Initial solution $S_{ini}$)
 2:     $S_{best} \leftarrow S_{ini}$
 3:     $HasImproved \leftarrow True$
 4:     while $HasImproved$ do
 5:         $HasImproved \leftarrow False$
 6:         for Each $Neig$ in $Neighborhood(V_{best})$ do
 7:             if $Evaluate(Neig) > Evaluate(S_{ini})$ then
 8:                 $S_{best} \leftarrow Neig$
 9:                 $HasImproved \leftarrow True$
10:             end if
11:         end for
12:     end while
13:     return $S_{best}$
14: end function
```

Line 6 of the algorithm uses the function $Neighborhood(Solution)$, that returns a set containing all possible solutions generated from $Solution$ by adding a single action to the current solution.

4.3 Tabu Search using the LP solution (TSLP)

The second applied technique was the Tabu Search algorithm [**glover1997tabu**], again using the Linear Programming solution as the initial candidate solution. In [**puchinger2010**], the authors claim that Tabu Search initialized with LP solution is the best known method to solve Multidimensional Knapsack Problem. We expect that the Tabu Search algorithm to be more prone to scape local minima, since it can accept solutions worst than the current one. The pseudocode is depicted in the algorithm 2 [**brownlee2011clever**].

We have also compared the solution quality if we randomly start the TS algorithm in the search space and found that in our experiments it never found a better solution than using the truncated Linear Problem solution as a start.

Algorithm 2 Tabu Search Algorithm

```
 1: function TABU SEARCH(Initial solution $S_{ini}$)
 2:     $S \leftarrow S_{ini}$
 3:     $S_{best} \leftarrow S$
 4:     $TabuList \leftarrow \emptyset$
 5:     while $StoppingCondition$ do
 6:         $CandList \leftarrow \emptyset$
 7:         for Each $S_{cand}$ in $TabuNeighborhood(S)$ do
 8:             if $featureDiff(S_{cand}, S) \notin TabuList$ then
 9:                 $CandList \leftarrow CandList + S_{cand}$
10:             end if
11:         end for
12:         $S \leftarrow BestCand(CandList)$
13:         if $Fitness(S) > Fitness(S_{best})$ then
14:             $TabuList \leftarrow TabuList + featureDiff(S, S_{best})$
15:             $S_{best} \leftarrow S$
16:             while $size(TabuList) > maxSize$ do
17:                 $RemoveOldest(TabuList)$
18:             end while
19:         end if
20:     end while
21:     return $S_{best}$
22: end function
```

The for loop on Line 7 uses the function $TabuNeighborhood(Solution)$, that returns a neighborhood of solutions generated from $Solution$. In our case, we chose to return in this list the same neighborhood of GALP concatenated with a randomly generated set of size n_{tabu} that differ from the base solution by at most three actions. For instance, if $Solution = \{10, 12, 30, 10\}$, a possible return of $TabuNeighborhood(Solution)$ could be $\{\{12, 11, 30, 10\}, \{10, 12, 29, 10\}, \{9, 11, 30, 10\}\}$.

5 Experiments

5.1 Experimental Setup

Computational Environment All experiments were run in Intel Core i5-2310 CPU @ 2.90GHz machines with 8GB of RAM memory running Linux Mint 13 64 bits. To facilitate the job distribution we have used the HTCondor framework.

Parameters The GALP algorithm has no settable parameters, the TSLP algorithm on the other hand, has 3 parameters: the number of iterations of the Tabu Search algorithm (10000), the size of the tabu list (1000) and the number of random neighbors per iteration ($n_{tabu} = 500$). The parameters were set up by empirically experimenting several different combinations.

Implementation To solve the integer programming problems we have used the GLPK LP/MIP solver [**GLPK**], version 4.43. We have implemented the heuristics using the Java programming language and executed/compiled the codes using the Java(TM) SE Runtime Environment (build 1.7.0_10-b18). All implementations are available at `http://ninfa.inf.ufes.br/ICTAI2013/impl.zip`.

5.2 Instance Generator

To create the instances for our analysis we have implemented a random instance generator. We have tried to mimic the characteristics of real-world actions supplied by the local EDCO. Algorithm 3 defines the pseudocode of the random instance generator.

Lines 2 to 6 present the generation of the yearly budgets from a Gaussian distribution (function $GaussRandom(mean, variance)$) that in turn has its variance and mean draw from a uniform distribution of fixed parameters (function $URandom(Lower\,bound, Upper\,bound)$). The parameters were inspired by real-world actions, given by the local EDCO.

Lines 7 to 12 generate the cost and market of each action, the cost comes from an uniform random distribution with fixed parameters. Note that in line 8 we use the Min function to guarantee that the action may performed at least one time. Line 11 defines the market of each action as a function of the total budget over all years ($Sum(o)$) and the cost of the action. That is, the cheapest the action the larger the market.

Lines 13 to 27 are used to build the reverse-ordered normalized weight matrix that is used to build the recuperation over the years. After line 27, the j-th line

Algorithm 3 Random instance generator

```
1: function GENERATOR(Correlation factor $\alpha$, Number of years $M$, Number of actions $N$)
2:     $BudgetMean \leftarrow Random_{Uniform}(700, 800)$
3:     $BudgetVariance \leftarrow Random_{Uniform}(10, 20)$
4:     for $i = 1 \rightarrow M$ do
5:         $o_{i1} \leftarrow Random_{Gauss}(BudgetMean,\ BudgetVariance)$
6:     end for
7:     for $j = 1 \rightarrow N$ do
8:         $c_{j1} \leftarrow Min(Min(o), Random_{Uniform}(1, 100))$
9:     end for
10:    for $j = 1 \rightarrow N$ do
11:        $m_j = 2Sum(o)/(M \cdot c_{j1})$
12:    end for
13:    for $j = 1 \rightarrow N$ do
14:        for $i = 1 \rightarrow M$ do
15:            $W_{ji} \leftarrow Random_{Uniform}(0, 1)$
16:        end for
17:        $W_j \leftarrow ReverSort(W_j)$
18:    end for
19:    for $j = 1 \rightarrow N$ do
20:        $s \leftarrow 0$
21:        for $i = 1 \rightarrow M$ do
22:            $s \leftarrow s + W_{ji}$
23:        end for
24:        for $i = 1 \rightarrow M$ do
25:            $W_{ji} \leftarrow W_{ji}/s$
26:        end for
27:    end for
28:    for $i = 1 \rightarrow M$ do
29:        for $j = 1 \rightarrow N$ do
30:            $e_{j,k} \leftarrow Max(0, Abs((1-\alpha) \cdot c_{j1} \cdot W_{ji} +$
31:                    $Random_{Uniform}(-100, 100) \cdot \alpha))$
32:        end for
33:    end for
34: end function
```

of matrix W will contain a vector of values that add up to 1 and are reversely ordered, e.g., a possible W matrix for 3 years and 4 actions could be:

$$W = \begin{pmatrix} 0.7 & 0.2 & 0.1 \\ 0.8 & 0.1 & 0.1 \\ 0.6 & 0.2 & 0.2 \\ 1.0 & 0.0 & 0.0 \end{pmatrix}.$$

Finally from lines 28 to 33 the electricity recuperation values are defined for each action and each year. In line 30 we may see that the use of matrix W guarantees that the electricity recuperation has a decreasing tendency over the years. In this line we also present the use of the correlation parameter $\alpha \in [0, 1]$. The closest α is to one, the weaker is the correlation.

The generator was implemented in the Python programming language and its implementation and random instances are available at `http://ninfa.inf.ufes.br/ICTAI2013/gen.zip`.

5.3 Time Analysis

The Impact of Correlation It has been demonstrated the classical Knapsack Problem is pseudo-polynomial in the size of the input [**garey1978**] and it is a established fact that a weak correlation among weight and value greatly reduces the difficulty of the instances [**david2005**]. Literature on the knapsack problem states that the difficult problems are the ones that exhibit a strong correlation between the value of the item and its weight.

To test if this phenomena also happens in our problem, we have tested the values $[0.0, 0.1, 1.0]$ for α, the parameter that controls the correlation of the value of the action in respect to its cost, in a number of problem sizes. When α is 0, there is a strong correlation between the cost of the action and its electricity return, that is, the more expensive the action, the more effective it is. When α is 1 there is no correlation between the value of the actions and its effectiveness. When α is 0.1, there is a weak correlation between cost and value. The α values of 0.0, 0.1 and 1.0 are equivalent to the following classes of problems defined by [**david2005**] (respectively): *subset sum instances, weakly correlated instances and uncorrelated instances.*

To measure running times we have generated instances for each considered α, varying both the amount of years and actions in the interval $[5, 15]$. For each triple defined by the value of α, the number of years and actions, we have generated 10 instances and measured their running time to find the exact solution. We have observed that the variance of the running times to solve these 10 instances was too large for the mean to be significative, so we have adopted a alternative strategy to compare problem sizes and α: we assume that running times of more than one hour are prohibitive for our application, since EDCO's usually test several portfolios to make their investment decision. So we consider that these instances failed to find the solution in practical times.

Figures 1 to 5.3 display the amount of instances that successfully executed in less than one hour, given a value of α. For each cell of the figure the exact algorithm was run 10 times using random instances of the problem. The closer to black, the greater the amount of successful runs. From the figures it is clear that the bigger the problem the more likely it is that it will take more than one hour to execute, since cells closer to $(15, 15)$ tend to be closer to white. Also, it may be observed that the more correlated the cost is with the profit (the smaller the α), the harder the problems seems to be, as predicted by the literature.

.png `\relax .png` .png `\relax .png` .png `\relax .png` [scale=0.73, trim=0.75cm 0cm 0 2cm, clip=true]imgs/hard.pdf

(b) Weak cor- re- la- tion be- tween cost and profit.

.png `\relax .png` .png `\relax .png` sub[scale=0.73, trim=0.75cm 0cm 0 2cm, clip=true]imgs/easy.pdf

No cor- re- la- tion be- tween cost and profit.

5.4 Solution Analysis - Easy Instances

In this subsection we compare the solution quality of the implemented heuristics in the set of easy instances, there is, the instances in which the exact algorithm found the optimal solution in less than one hour. We have also limited the running time of the heuristics in one hour.

Figures 3 to 8 display the average ratio between the optimal solution (when available) and the solution found by the heuristics. Small relative ratios are darker than large differences. A ratio of 1 means that the heuristic has found a solution with the same quality than the exact algorithm. When no solution was found by the exact approach in less than one hour for a tripe $(\alpha, years, actions)$, we display the value "*na*" in the corresponding cell.

Figures 3 to 8 show that all heuristics managed to find very close solutions to the exact (less than 1% difference). Also, the paler aspect of the TSLP figures (specially when $\alpha = 0.0$) suggests that it managed to beat the GALP algorithm considering solution quality. In fact, considering only these easier problem sizes, the paired Wilcoxon signed-rank test [**japkowicz2011evaluating**] rejects the null hypothesis that the algorithms are the same with a p-value of less than 10^{-11}.

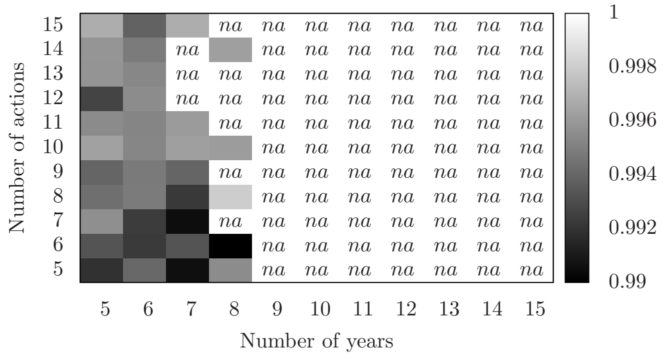

Fig. 1: Comparison between the optimal solution (when available) and the solution by the GALP for $\alpha = 0.0$.

5.5 Solution Analysis - All Instances

In this subsection we compare the behavior of the metaheuristics in respect to one another considering all instances. We cannot compare the results with the optimal solution since they are unknown for the harder instances.

Figures 9 to 11 display the relative distance between the two proposed heuristics considering the solution quality for the three studied correlation values. Each

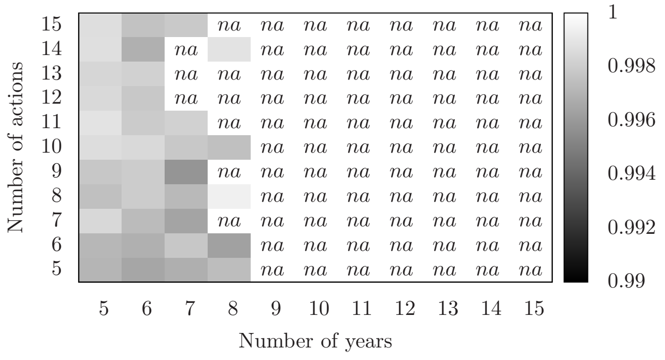

Fig. 2: Comparison between the optimal solution (when available) and the solution by the TSLP for $\alpha = 0.0$.

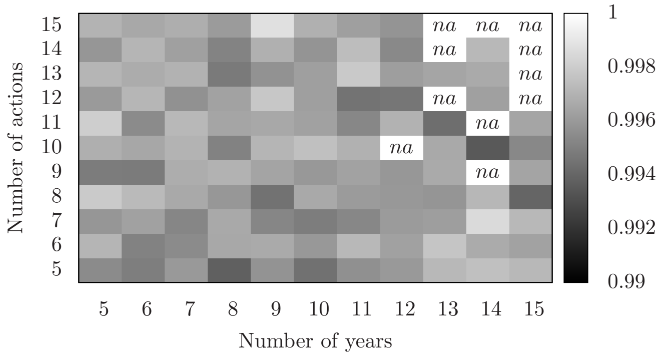

Fig. 3: Comparison between the optimal solution (when available) and the solution by the GALP for $\alpha = 0.1$.

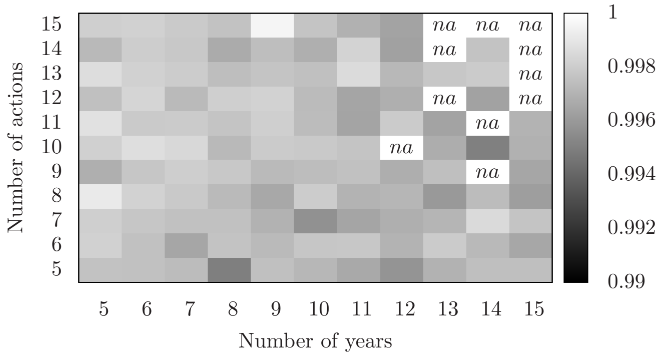

Fig. 4: Comparison between the optimal solution (when available) and the solution by the TSLP for $\alpha = 0.1$.

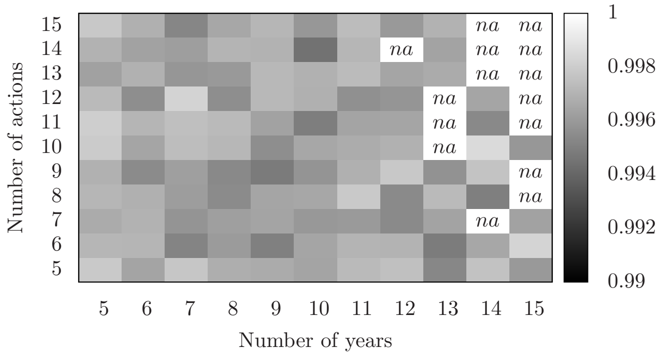

Fig. 5: Comparison between the optimal solution (when available) and the solution by the GALP for $\alpha = 1.0$.

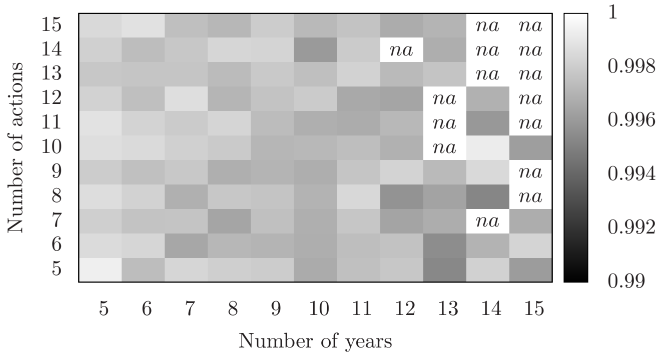

Fig. 6: Comparison between the optimal solution (when available) and the solution by the TSLP for $\alpha = 1.0$.

cell contains the mean ratio between the quality of the GAPL and TSLP algorithms, here the mean is meaningful because the values of the ratio have a reasonable variance.

From the scale of the figures (varying between 1 and 0.993 at most) it is clear that both algorithms achieved very similar results, however the TSLP algorithm managed to outperform the GAPL, in average, in every cell of the figures (no value is greater than 1). However GALP did manage to outperform the TSLP algorithm in some instances of the problem, so we perform the paired Wilcoxon signed-rank to confirm that the TSLP is indeed superior to the GALP algorithm. And again we may reject that null hypothesis that the algorithm are equal with a p-value of less than 10^{-11}.

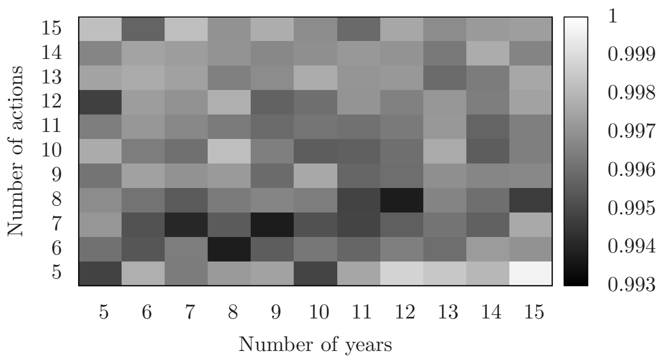

Fig. 7: Relative distance between the two proposed heuristics for $\alpha = 0.0$.

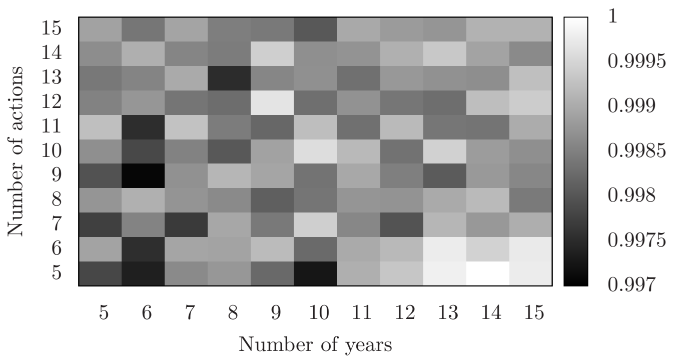

Fig. 8: Relative distance between the two proposed heuristics for $\alpha = 0.1$.

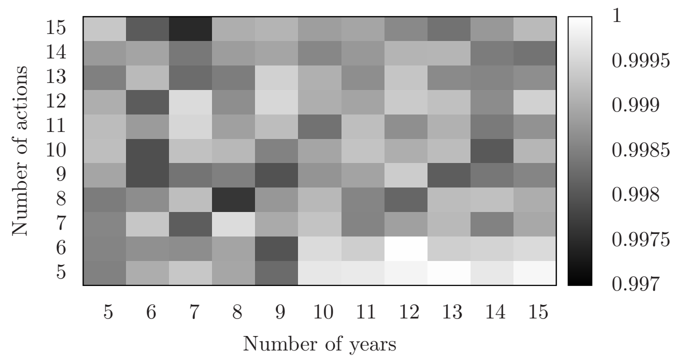

Fig. 9: Relative distance between the two proposed heuristics for $\alpha = 1.0$.

Figures 12 and 13 display the histogram of the ratio of the solution found by the heuristic algorithms and the optimal solution. From the histograms we can conclude that TSLP has a better behavior than GALP, showing a smaller variance and larger ratio mean.

Although the time limit was set to one hour, the TSLP had a maximum running time of less than one minute, with an average time of 20 seconds, and GALP had a negligible running time for all instances.

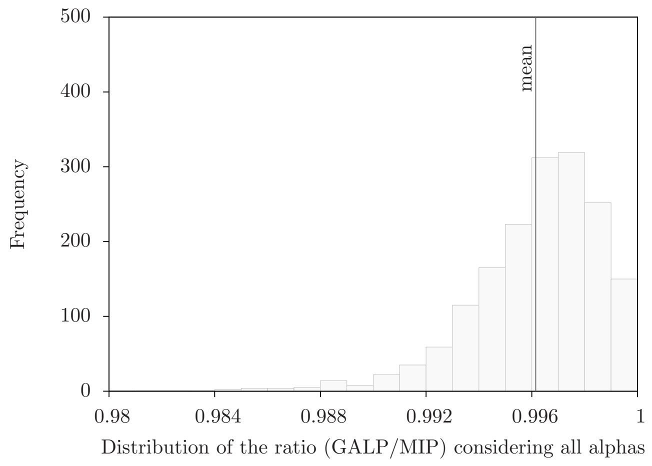

Fig. 10: Histogram of the ratio of the solution found by the GALP heuristic and the optimal solution.

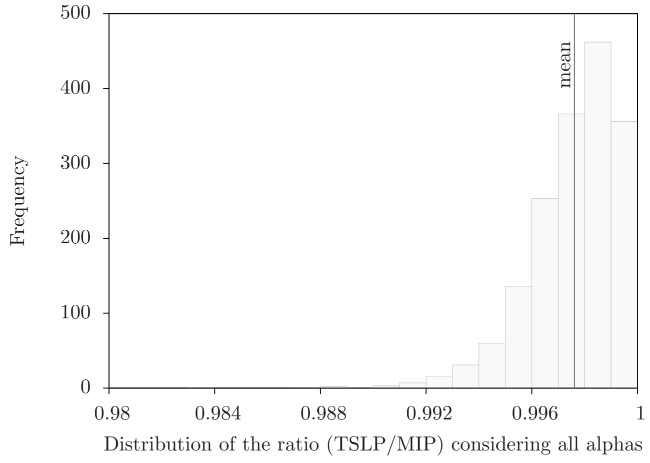

Fig. 11: Histogram of the ratio of the solution found by the TSLP heuristic and the optimal solution.

6 Conclusion and Future Work

This works presents a modeling of a relevant problem to Electricity Distribution Companies in developing countries. The modeling yields a challenging optimization problem. We propose a exact technique to solve easier instances of the problem and two heuristics to tackle the more difficult instances.

We conclude that our incarnation of the classical Knapsack is sensible to the correlation between weight and cost, as predicted by the literature on the problem.

We have tested two heuristic algorithm, namely Tabu Searh using the Linear Problem solution as an initial search point (TSLP) and Gradient Ascent using the Linear Problem solution as an initial search point (GALP), both achieving good solutions and execution times. In particular, the TSLP algorithm was statically better than the GALP algorithm in our test instances, with bigger running times.

Future work includes the investigation an algorithm for the complete version of the problem and a deeper study of what makes some instances take more time than others despite having the same amount of years, actions and α and being randomly generated in a similar way.