Wellington Rodrigo Monteiro

A large-scale hybrid optimization strategy formulti-objective problems

Curitiba

2018

Wellington Rodrigo Monteiro

A large-scale hybrid optimization strategy formulti-objective problems

Dissertation document presented to the Industrial & Systems Engineering Graduate Program, System Optimization, Modeling and Control research group from the Pontifical Catholic University of Paraná as a partial requirement for the Master's degree in Industrial & Systems Engineering.

Supervisor: Gilberto Reynoso Meza

Curitiba

2018

Dados da Catalogação na
Publicação Pontifícia
Universidade Católica do
Paraná
Sistema Integrado de Bibliotecas –
SIBI/PUCPR Biblioteca Central
Luci Eduarda Wielganczuk – CRB 9/1118

M775L
2018

Monteiro, Wellington Rodrigo
A large-scale hybrid optimization strategy for multi-objective problems / Wellington Rodrigo Monteiro ; supervisor: Gilberto Reynoso Meza. – 2018.
111 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2018
Bibliografia: f. 105-111

1. Engenharia de produção. 2. Algoritmos genéticos. 3.Otimização combinatória. I. Meza, Gilberto Reynoso. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Engenharia de Produção e Sistemas. III. Título.

CDD 22. ed. – 670

Abstract

In the optimization field it is common to find algorithms originally capable of addressing problems with a single objective (such as direct search methods including Nelder-Mead simplex method, Hooke-Jeeves and pattern search) and other strategies known to address multi-objective problems (multi-objective evolutionary algorithms, for instance). The latter are able to solve multi-objective problems with varying amounts of variables and constraints. However, these strategies do not adapt very well to some real problems that have a high amount of variables. It happens due to their low performance combined with a high resource usage as well as sometimes being unable to reach and generate an adequate solution set. Considering these reasons, this research proposes a new hybrid (combining a genetic algorithm approach with local search) optimization strategy for multi-objective continuous problems having such characteristics while keeping an acceptable performance to be deployed in industrial applications such as in the meat industry.

The proposed hybrid strategy is intended to have lower computer resource usage through parallelization and new parameters to be used in order to better use these resources as needed. By computer resources, it is understood as the memory and processor usage. For example, in the proposed strategy, some parameters enable the reduction on the number of solutions being transferred from one generation to the other or the number of solutions being evaluated in the local, stochastic search algorithm. With these strategies, the proposed strategy aims to allow the evaluation of problems containing a large number of variables (i.e. containing tens of thousands of variables or more). Other than this, it also must be able to attempt to quickly reach the real Pareto front and be easy to evaluate, adapt and port.

This strategy originated from the situation of a large meat company which used a single objective optimization algorithm to generate the production plan for its slaughterhouses, being the objective the maximization of the theoretical profits. However, this strategy did not consider the reliability of each slaughterhouse — each slaughterhouse covered only partial quantities of the proposed production plan determined by the algorithm and, therefore, the profits informed by the algorithm differed from the real values. As such, a multi-objective optimization algorithm could address the scenario considering the maximization of the reliability of the overall plan (determined from the reliability rates of the slaughterhouses) as another objective. While this algorithm was executed and provided viable results, it used too much computer resources and took too long to completely run due to the large number of variables in the problem.

As a feedback of these outcomes, the need to create a multi-objective optimization algorithm

with low resource usage and with faster results arose. Also, since the company found some opportunities to improve the accuracy of their own data, the new strategy was compared against two other algorithms using other test problems as complex as the original one, comparing both their computer resource usage (memory and processor) as well as the hypervolume for the Pareto front approximations and the approximations in themselves.

Keywords: hybrid optimization. multi-objective optimization. many-variable optimiza- tion.

Resumo

Na área de otimização é comum encontrar algoritmos originalmente capazes de endereçar problemas com somente um objetivo (como o Nelder-Mead ou o Hooke-Jeeves) e outras estratégias conhecidas por endereçar problemas multi-objetivo (por exemplo, algoritmos evolucionários multi-objetivo). Estas últimas são capazes de solucionar problemas multi- objetivo com diferentes grandezas de variáveis e restrições. Contudo, estas estratégias não se adaptam muito bem para problemas reais que possuem um grande número de variáveis. Isto ocorre por conta do seu baixo desempenho combinado com um alto uso de recursos do computador e até mesmo não conseguir alcançar um conjunto adequado de soluções ocasionalmente. Considerando estas razões, esta pesquisa propõe uma nova estratégia de otimização híbrida (isto é, combinando um algoritmo genético com busca local) para problemas contínuos multi-objetivo que possuam estas características enquanto mantém uma performance aceitável para ser utilizado em aplicações industriais tais como aquelas na agroindústria.

Espera-se da estratégia híbrida proposta um baixo uso de recursos computacionais através do emprego do paralelismo e de novos parâmetros a serem usados para melhor utilizarem estes recursos conforme necessário. Por recursos computacionais, compreendem-se os usos da memória e do processador. Por exemplo, na estratégia proposta, alguns parâmetros possibilitam a redução no número de soluções sendo transferidas de uma geração a outra ou então no número de soluções sendo avaliadas pelo algoritmo de busca local estocástica. Com estas estratégias, a estratégia proposta busca possibilitar a otimização de problemas contendo um grande número de variáveis (contendo dezenas de milhares de variáveis ou mais, por exemplo).

Esta estratégia originou-se de uma situação de uma grande empresa da agroindústria a qual utilizava um algoritmo de otimização mono-objetivo para gerar o plano de produção para os seus frigoríficos, sendo o objetivo a maximização dos possíveis lucros. Contudo, esta estratégia não considerava a confiabilidade de cada frigorífico — cada frigorífico atendia apenas à uma quantidade parcial daquilo que era sugerido pelo plano de produção determinado pelo algoritmo e, portanto, os lucros informados pelo algoritmo eram diferentes dos valores reais. Assim sendo, um algoritmo de otimização multi-objetivo poderia endereçar este cenário considerando esta confiabilidade como um segundo objetivo. Enquanto que um algoritmo multi-objetivo tenha sido executado e fornecido valores viáveis, ele utilizava muitos recursos computacionais e demorou muito tempo para completar o seu funcionamento devido ao grande número de variáveis no problema.

Como um resultado dessas situações, nasceu a necessidade de criar um algoritmo de

otimização multi-objetivo de baixo consumo de recursos computacionais e que fosse capaz de gerar resultados mais rapidamente. Ademais, uma vez que a empresa encontrou algumas oportunidades de melhora na acurácia dos seus próprios dados por conta desta pesquisa, a nova estratégia foi comparada com outros dois algoritmos utilizando problemas matemáticos cujas complexidades assemelhar-se-iam à complexidade do problema do plano de produção, comparando tanto o uso de recursos computacionais (memória e processador) como os hipervolumes das aproximações das frentes de Pareto bem como as aproximaçõesem si.

Palavras-chave: otimização híbrida. otimização multi-objetivo. otimização multivariável.

List of Figures

List of Tables

List of abbreviations and acronyms

AOF	Aggregate Objective Function
CCGA	Cooperative Coevolutionary Genetic Algorithm
CCGDE3	Cooperative Coevolutionary GDE3
CPSO	Cooperative Particle Swarm Optimization
CPU	Central Processing Unit
DM	Decision-making
EMO	Evolutionary Multi-objective Optimization
ERP	Enterprise Resource Planning
GA	Genetic Algorithm
GFCL	Generate-First Choose-Later
GDE3	Generalized Differential Evolution, 3rd version
MA	Memetic Algorithm
MCDM	Multi-criteria Decision-making
MOEA	Multi-objective Evolutionary Algorithm
MOO	Multi-objective Optimization
MOOD	Multi-objective Optimization Design
MOP	Multi-objective Problem
NIS	Negative Ideal Solution
NNC	Normalized Normal Constraint
PIS	Positive Ideal Solution
PIT	Practically Insignificant Tradeoff
PROMETHEE	Preference Ranking Organization METHod for Enrichment of Evaluations
RAM	Random Access Memory

SaNSDE	Self-adaptive Neighbourhood Search Differential Evolution
SO	Single-objective
sp-MODE	Multi-objective Differential Evolution with Spherical Pruning
TOPSIS	Technique for Order of Preference by Similarity to Ideal Solution

Contents

Acknowledgements

Thanks to the Pontifical Catholic University of Paraná, specially to its Polytechnic School and its Industrial & Systems Engineering Graduate Program (PPGEPS).

Thanks to Prof. Gilberto Reynoso-Meza for his help, assistance and overwhelming support.

Special thanks to the family, friends and wife of the author for their continuous support, patience and care.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Introduction

The concept of *optimization*, by itself, is simple. From a pragmatic standpoint, it revolves around the idea of *improving* something and choosing the best alternative (or alternatives) - be it a process or a product, for example Stadler (2013). Independently of what it might be, it has one or multiple constraints and variables according to the context. For example, improving a given production plan might result in multiple values representing the production distribution across multiple production lines and/or plants for a set of products in different dates. These variables might even be numeric or binary. One important notion here is that both the concepts of improvement and optimization are also associated with the idea of *quality* - after all, if something is improved it should also have better quality as a result. Quality, as is, is a relative concept since it depends from the standpoint of a person and also the context, the timing, and so on. Therefore, when something - named *problem* from now on - must be improved, it must be defined in an exact, mathematical fashion. From it, it is possible to determine what exactly should be improved and under what set of criteria it must comply.

Recalling the example of the production plan to be improved, it is possible for one to assume that, from its description as is, a given optimization algorithm should provide as a result the *best* production plan. However, for a plant maintenance coordinator the word *best* might mean that the resulting plan will have separate, equilibrated maintenance windows along the day. For the logistics manager it might imply in a plan that generates the least stock in plant. For the Sales department it might imply in a plan with the most profitable material, even though it might generate a lot of waste since the least profitable raw material or semi-finished goods would be simply ignored. From this quick example it is possible to identify that the result of the algorithm shall be oriented to the definition and understanding of the objective according to the one in charge of designing it - i.e. the *problem designer* must define exactly what he needs in the end. This task must encompass not only the objective itself, but also all the limitations and situations to be considered. In short, it means that the problem designer must carefully define the entire model since even small errors might produce largely distinct results.

While this situation is easy to understand, in most real-world scenarios the degree of difficulty may rise exponentially depending on the case. The first issue is within the data collection itself since the data is not always ready for use or even it is either nonexistent or unreliable - this situation is also seen in other industry frameworks and techniques where a reliable data source is required, such as Lean Six Sigma, as seen in Albliwi et al. (2014). Secondly, the scenarios and situations can reach the hundreds or thousands - in the production plan example, the distribution of 10 different products across four weeks

for eight plants and with differing plant capacities and market demands along the weeks can already reach hundreds of variables. Data sources such as *common optimization for radiation therapy (CORT)* dataset Craft et al. (2014) in the radiation therapy field or housing-related datasets as the example shown in Hallac, Leskovec and Boyd (2015) and engineering-related problems such as the ones in Liao et al. (2014) are real-world, complexcases that meet the criteria explained above.

Up to this point, there are already some challenges to be tackled: 1) the difficulty in retrieving and organizing reliable data; 2) the difficulty in designing the problem and its constraints; 3) the large number of variables and constraints to be handled in some real-world problems; and 4) the computational performance of the optimizers involved in resolving said problems. Focusing in the latter two items since they are more interesting from a computing standpoint, it is important to highlight the myriad of single-objective algorithms with varying strategies and, therefore, differing performance depending on the algorithm chosen. Now, when many objectives are involved - that is, when a combination of objectives are looked for with implicit tradeoffs between them - the complexity is even higher. Back on the production line example, one may search not only for the most profitable solution, but instead may look for the solution with the most profits *and* also with the least stock room space used. In such scenarios there should not have only one solution since the one with the most profits may as well use more stock than other solutions, and vice-versa. Therefore, a *list* of possible solutions is generated instead so the problem designer can choose one out of it that may best suit his needs. This list should be comprehensive but also short and diverse at the same time, thus bringing another layer of complexity to the process. When dealing with real-world, large scenarios, most algorithms may use too much processing power - and, therefore, time - attempting to find the bestsolution.

For this reason, there is a strategy named *hybrid optimization* Juang (2004), which attempts to accelerate the discovery of the best solutions by forcing improvements within each step - let us consider that the discovery of a set containing the possible solutions is achieved after 1000 iterations where each iteration generates a set and the next iteration tries to improve the set created in the past iteration. This strategy, as shown in more detail in this document, employs different methods which, in the long run, might reduce the quantity of iterations and/or time required to reach the *best* set of solutions as well as the required computational time to achieve that which is specially interesting in complex, real- world problems. For example, a genetic algorithm (GA) might generate better and better solutions each generation through tournament, mutation and recombination techniques. The GA is an example of a metaheuristic optimization algorithm since there is no need to deeply customize and adapt it depending on the problem, as seen in BoussaïD, Lepagnot and Siarry (2013). A case of a *hybrid optimization* modification to this algorithm would include an additional step after the end of each generation where some variables of these

solutions are randomly changed in order to find even better solutions and using them instead.

Therefore, in order to better address the situations noted above there is a need to implement a mix of techniques in order to attempt to find quickly the best set of solutions with the least side-effects as possible - that is, in the hurry of finding the best solutions the algorithm might ignore some of them that would be considered otherwise. Having that said and considering the big picture explained above, this research proposes a hybrid algorithm targeted for large-scale, multi-objective problems. Large-scale, as seen in Cheng et al. (2016), are problems with at least hundreds or thousands of decision variables. Multi-objective problems, as shown in Huang, Gu and Du (2006), are problems with two or more solution objectives. Since this algorithm was originally targeted to be used in the meat industry - more specifically on the generation of production plans - it was developed with integer variables in mind (i.e. without considering discrete variables). This document is divided five chapters besides the current one, as follows: the current chapter is the introduction to the research itself. After it, the first numbered chapter lays ground to the background used by the algorithm, explaining what exactly is understood by *single-objective and multi-objective optimization* (i.e. the optimization of problems with either only one objective or problems with two or more objectives), *large-scale optimization* (i.e. the optimization involving at least thousands or tens of thousands of variables to be considered) or the already existent hybrid techniques, for example. It is important to explain the logic behind single-objective optimization since many of its concepts are inherited by multi-objective optimization, which is the target of this research. The following chapter shows the preliminary contributions of this research. The next chapter revolves around the proposal of the expected deliverables for this research while the following chapter presents the a discussion of the results of this proposal. The next chapter shows the conclusions of this research while the final chapter lists the bibliography used.

1 BACKGROUND

Before understanding the new algorithm proposed in this document it is of paramount importance understand where and how it is applicable. The new algorithm is intended to be used in multi-objective problems - therefore, it is worth understanding what determines a given problem as *single-objective* or *multi-objective*. Going one step further, it is also important to identify all the steps that compose a multi-objective algorithm as well as some of its implementations - the multi-objective evolutionary algorithm (MOEA), for example, is one of the stepping stones of the new algorithm. In the same way, it is relevant to understand all the tests (problems) that will be subject to the new algorithm as well as other algorithms for comparison purposes.

1.1 Single-objective Optimization

From a computing standpoint, single-objective (SO) algorithms aims to find the values for a given equation that provide the highest - or lowest - value. In this perspective, the equation is the mathematical representation of the problem to be improved which result is the objective itself - when the objective should be minimized, the lowest the result is, the better. Analogously, when the objective should be maximized, the highest the result is, the better. The problem may be represented as follows, as seen in Rao (2009):

$$Find\ \mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \ which\ minimizes\ F(\mathbf{X}) = \mathbf{Y} \qquad (1.1)$$

In this case, x_i, $i = 1, 2, ..., n$ are the *design variables* of a problem, where each variable is a characteristic of the problem. In the production plan example, each variable can represent the production of a given material for a given plant, for a given time period. These variables combined form what is called *design vector*. The design vector is used in a *objective function* F (**X**) which generates a result. Using the same example, each design variable may have a value assigned to them that represents their cost of production - there might be different costs of production depending of the combination of product, plant and time. If the problem designer wants to minimize the costs of production as much as possible then this objective function will be a minimization problem. On the other hand, he might instead assign values associated with the estimated profits. Then, due to its nature (the designer wants the highest profit), the problem would be one of maximization

instead. Most techniques, since they are oriented to minimization, commonly require the problem designer to convert the problem from maximization to minimization by inverting both the problem and its constraints. Due to this reason, from now on only references to minimization problems will be considered except when explicitly noted otherwise.

The constraints are a list of equations and/or inequations the objective function must comply. These constraints define limits for the algorithm in a way that only reasonable solutions can be achieved. As also noted by Rao (2009), these constraints are defined as:

$$g_j(\mathbf{X}) \leq 0, j = 1, 2, ..., m \tag{1.2}$$

$$l_j(\mathbf{X}) = 0, j = 1, 2, ..., p \tag{1.3}$$

where m and p can be different than n and/or themselves. The production plan problem can state as constraints the maximum capacity of each plant, the minimum production required for each plant, the market minimum and maximum requirements for each week and so on.

Therefore, the objective is always to find the *best* value - i.e. maximize or minimize as much as possible - for a given problem that is subject to a set of constraints that determine the area of feasible solutions. The concept of *area*, shown in the Figure 1, can be explained in the following way: considering a n-dimensional Cartesian space - henceforth named *design space*, where each axis represent one design variable, a set of infinite solutions can be represented along it. However, the constraints limits the space where the solutions are feasible - thus determining what is called as *feasible region*. From the set of feasible solutions, the one that outperforms the others in respect to the objective function is what is named the *optimum point* (or points, if there is more than one point with the same minimum).

One challenge found by the algorithms in charge of resolving SO problems is the local optimum which does not always equal to the global optimum. Therefore, an algorithm may not always guarantee that it found the global optimum instead of a local optimum. In other words, the minimum value found by the algorithm might be only the minimum value in a region of the feasible space instead of the minimum value for the feasible space as a whole. On the other hand, some algorithms might as well be complex for simpler problems. Due to this reason there are several SO algorithms available using different implementations such as Nelder-Mead, seen in Klein and Neira (2014); DIRECT, shown in Hao et al. (2016); Multidirectional Search, noted in Torczon (1997); Hooke-Jeeves, as seen in Lin, Ma and Cooper (2016) or Implicit Filtering, mentioned in Kelley (2017) techniques, among others shown, for example, in Kelley (1999).

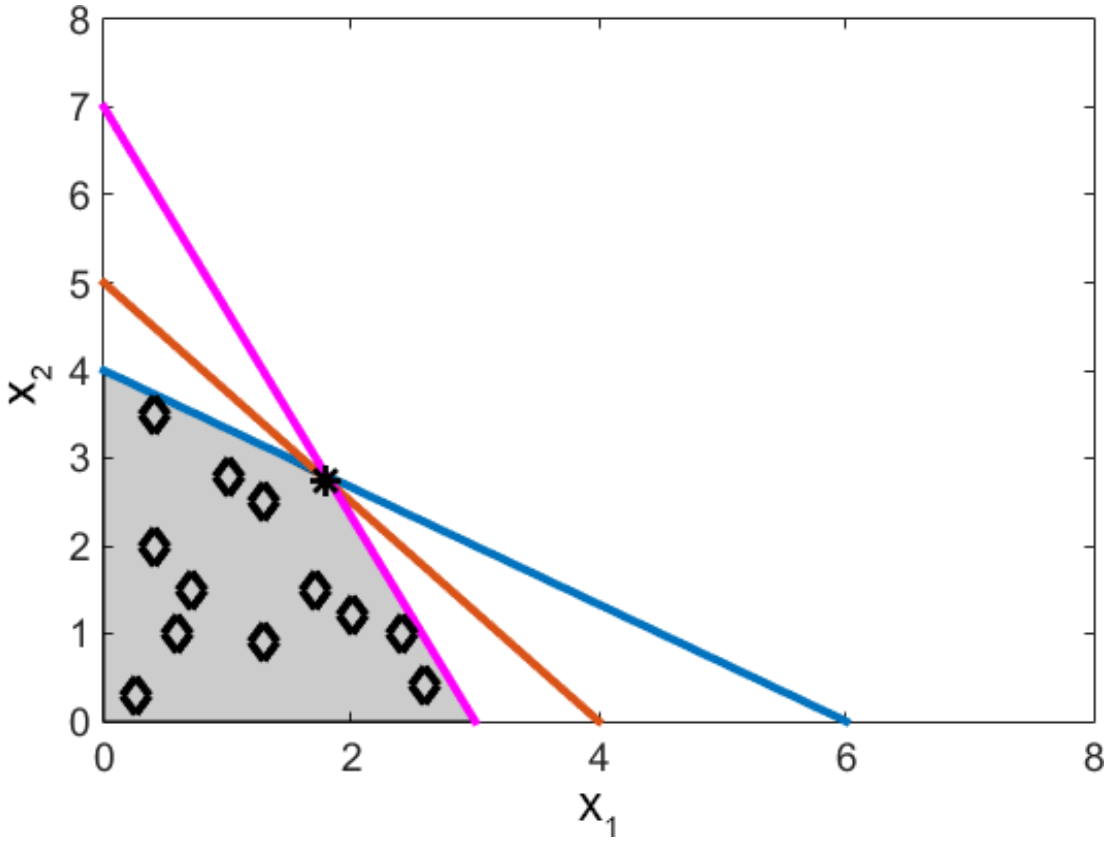

Figure 1 – A two-dimensional design space - each axis represent a design variable. The darker area is the feasible region where infinite feasible solutions are inside it, with some of them represented by the black diamonds. The optimum point is represented by the black asterisk.

1.2 Multi-objective Optimization

The general implementation behind the Multi-objective Optimization (MOO) differs from the SO algorithms due to its increased complexity. While the SO strategies only have one resulting value - meaning therefore that the problem designer will have one single "target" to look for, the multi-objective problems (MOP) have two or more objectives to be analyzed as seen in the equation (1.4) from Huang, Gu and Du (2006). In this equation, represented by f_1 and f_2 represents two different objectives where both are subject to the restrictions h_i and g_j. The Figure 2 shows an example of a problem with two objectivesand two variables.

$$\begin{aligned} \textit{Minimize} \quad & f(\mathbf{X}) = \{f_1(\mathbf{X}), f_2(\mathbf{X}), \ldots, f_n(\mathbf{X})\}, \\ & h_i(\mathbf{X}) = 0, i = 1, 2, \ldots, l, \\ \textit{Subject to:} \quad & g_j(\mathbf{X}) = 0, j = 1, 2, \ldots, J, \\ & \mathbf{X}^U{}_k \geq \mathbf{X}_k \geq \mathbf{X}^l{}_k k = 1, 2, \ldots, K \end{aligned} \tag{1.4}$$

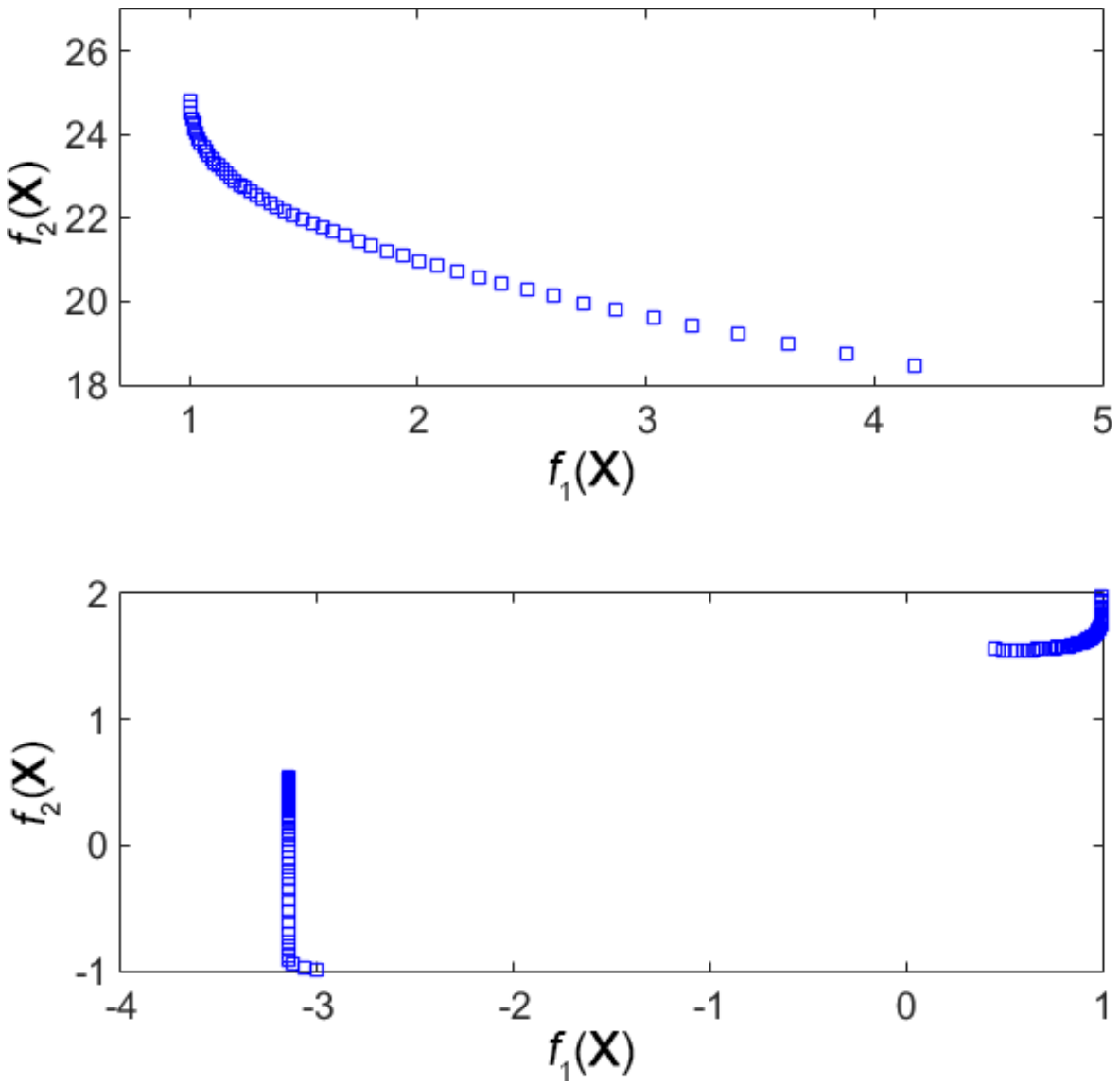

Figure 2 – The same solutions are shown in both charts. The problem has two objectives and two variables. The bottom chart, commonly used in SO strategies, shows the values found from the variables standpoint while the upper chart shows the values relative to the objectives. For MOPs it is more common to represent the solutions from the objective standpoint since the decision maker must, generally, analyze the tradeoffs between the multiple objectives first.

Back in the production plan example, consider the situation where the requirement is to minimize the costs of production and maximize the profits *at the same time*. Depending on the strategy, an infinite set of feasible solutions might be generated and all of them could be considered as the best solution - one of these solutions might have the best profits, but it would also be one of the costliest. Alternatively, it would be the cheapest production plan, but its profits would be somewhat low. Additionally, there would be other solutions falling in the between. For this reason, the problem designer would need to *choose* the best solution for his current needs by analyzing the tradeoffs between all the proposed solutions.

This way, the algorithm implementations for MOPs differ from the SO algorithms.

1.2.1 Fundamentals

As shown in Paula et al. (2017), there are two main strategies when resolving a MOP: *generate-first choose-later* (GFCL) and *aggregate objective function* (AOF). The AOF approach merges all the design objectives, commonly through weighting vectors that determine the relative importance of each objective. In this scenario, the problem designer
- or decision maker - must define the tradeoffs before the execution of the algorithm, generating as a result a single value similar to the SO problems. Therefore, distinct decisions could be generated depending on the weights used at the start of the process.

In order to understand the concept behind the GFCL approach it is important to also understand the concepts of Pareto optimality, Pareto set and Pareto front. As explained by Veldhuizen and Lamont (1998), a given solution (i.e. vector) v is said to be*Pareto dominated* by another solution u if u is partially less than v such as:

$$\forall i \in \{1, ..., p\}, u_i \leq v_i \wedge \exists i \in \{1, ..., p\} : u_i < v_i \tag{1.5}$$

If a solution is non-dominated when compared against all the other solutions it is said that it is a *Pareto optimal* solution. A set of these solutions is part of the *Pareto front*. Yet, an important detail should be noted: it is possible that a given set of non-dominated solutions generated by an algorithm are not exactly Pareto optimal because the real Pareto front was simply not found by the algorithm. Then, as mentioned in Reynoso-Meza (2014), these solutions are part of what is called the *approximated Pareto front* instead. In fact, this is not a great issue most of the times simply because the real Pareto front is unknown. As shown in the Figure 3, where each axis represent one objective, five solutions were found by a given algorithm. In the feasible solution space there were two dominated solutions - one by the solution A and another by the solution B. Therefore, they were not considered as part of the Pareto front. The solutions A, B and C, however, were the non-dominated solutions found by the algorithm and, for this reason, were part of the Pareto approximated front. In practice, B and C are not part of the real Pareto front but, since the algorithm could not reach it, they are part of the Pareto approximated front instead. Since different algorithms can reach different approximated fronts, it is important to attempt to reach as close as possible to the real front.

1.2.2 Algorithm Rationale

When using the GFCL strategy, the multi-objective optimization design (MOOD) does have at least three steps as shown in Reynoso-Meza et al. (2014): the multi-objective

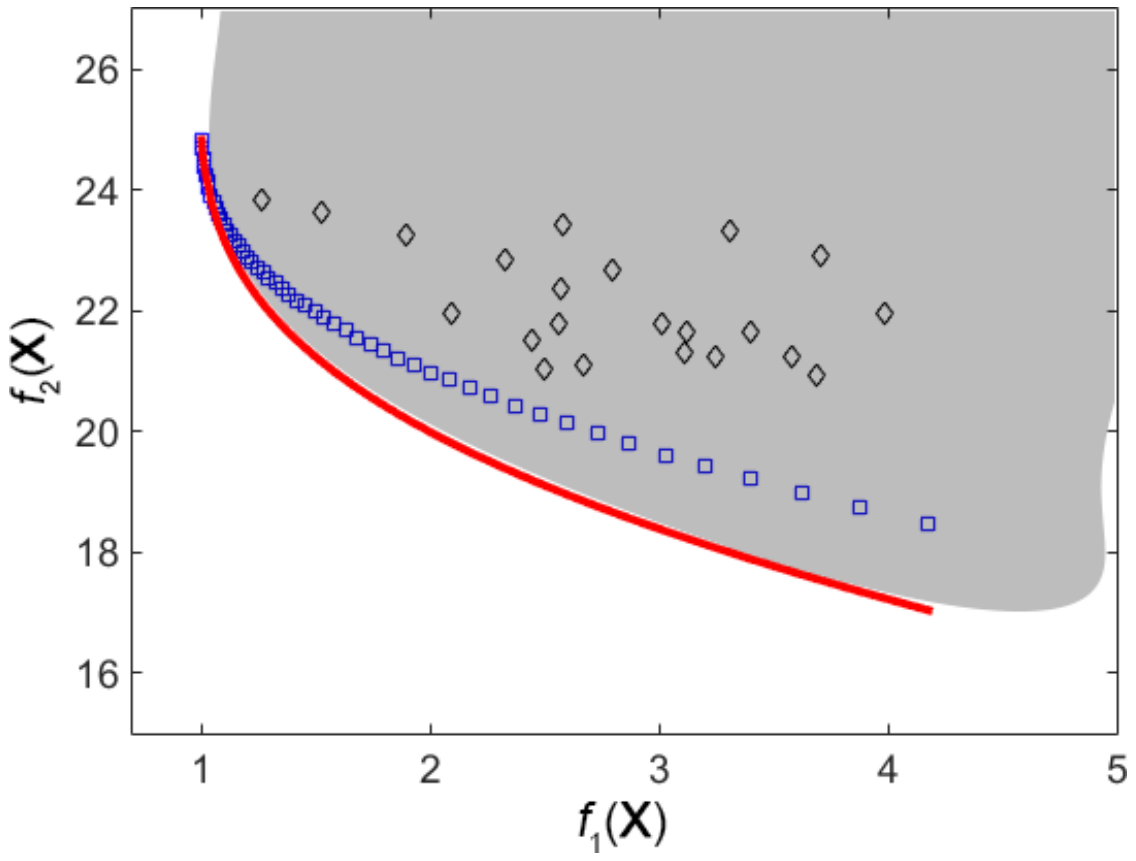

Figure 3 – Example of the Pareto optimality for a two-objective problem as shown by Reynoso-Meza (2014). The continuous line represent the Pareto optimal front. The dark area represents the feasible solution space. The diamonds represent feasible solutions dominated by other non-dominated solutions (represented by the squares). The squares overlapping the continuous line are non-dominated and Pareto optimal solutions while the other squares are simply non-dominated solutions.

problem (MOP) definition, the multi-objective optimization (MOO) and the multi-criteria decision making (MCDM).

The **MOP** definition is the step where the objective function and the constraints are defined. Considering that many objectives are involved, the quantity of variables and equations involved in the problem can be hard to manage. For this reason, sometimes the designer must also balance human readability and algorithm readability - i.e. sometimes, a problem that is easily understandable by a human might not be easily interpreted by the algorithm and vice-versa.

The **MOO** process comprises the computational efforts related to solving the problem itself and finding useful solutions. As shown in Miettinen (1999), there are some classical algorithms to achieve this, such as the weight sum and goal attainment techniques apart of a brute-forced one. The brute-force technique basically generates a given number of random-generated solutions. Then, between them the decision maker chooses the non-

dominated ones as part of the approximated Pareto front. Due to its simplicity it is more effective in simpler, bidimensional or tridimensional search spaces. On the other hand, since the chances of reaching the real Pareto front increases with the quantity of solutions generated at the same time that the percentage of discarded solutions - and, therefore, wasted computing power - also increases, it is not recommended to use it against larger search spaces, principally when parting from the assumption that an infinite number of solutions can be held within the space.

The weighted sum technique works as follows: considering that a general MOP canbe defined as:

$$Minimize\ F(\mathbf{X}) = [F_1(\mathbf{x}), F_2(\mathbf{x}), ..., F_k(\mathbf{x})]^T \tag{1.6}$$

and that is subject to:

$$g_j(\mathbf{x}) \leq 0; j = 1, 2, ..., m \tag{1.7}$$

where $F(x)$ is a vector of k objective functions that is subject to m inequality constraints, the composite objective function can be represented as seen in Naidu, Mokhlis and Bakar (2014):

$$U = \sum_{i=1}^{k} w_i F_i(\mathbf{x}) \tag{1.8}$$

where w_i is a weight assigned to each objective. Therefore, the designer must assign a bias for each objective before running the algorithm. As a result, the Pareto front generated will be biased depending on the weights informed as well as their relation between themselves.

The goal attainment approach, on the other hand, as informed in Fonseca, Fleming et al. (1993), implements goals such as that the general MOO equation (1.6) is modified by implementing a set of design goals g_i, each of them associated to each objective available such that:

$$f_i - w_i\lambda \leq g_i \tag{1.9}$$

where the minimization of the scalar λ leads to the finding of the specified goals either under or over attained in relation to the specified goals. Additionally, the weights w_i are weights and with values equal or above zero, also defined by the designer. Therefore, this set of design goals represent the expected Pareto optimal solution the designer is looking for.

Another strategy is called *Normalized Normal Constraint* (NNC) Messac, Ismail-Yahaya and Mattson (2003) and it is defined of the following steps:

Algorithm 1: Normalized Normal Constraint **Data:**
constraints, objective function, design space
Result: the Pareto front approximation
1 define the anchor points (i.e. the best possible points considering only one of the objectives);
2 form an Utopia line linking the anchor points;
3 divide the Utopia line into segments. The number of segments is defined by the designer;
4 **for** *each segment* **do**
5 evaluate *n* positions;
6 apply a dominance filter to the current segment;
7 **end**
8 apply a *PIT* (practically insignificant tradeoff) if needed.

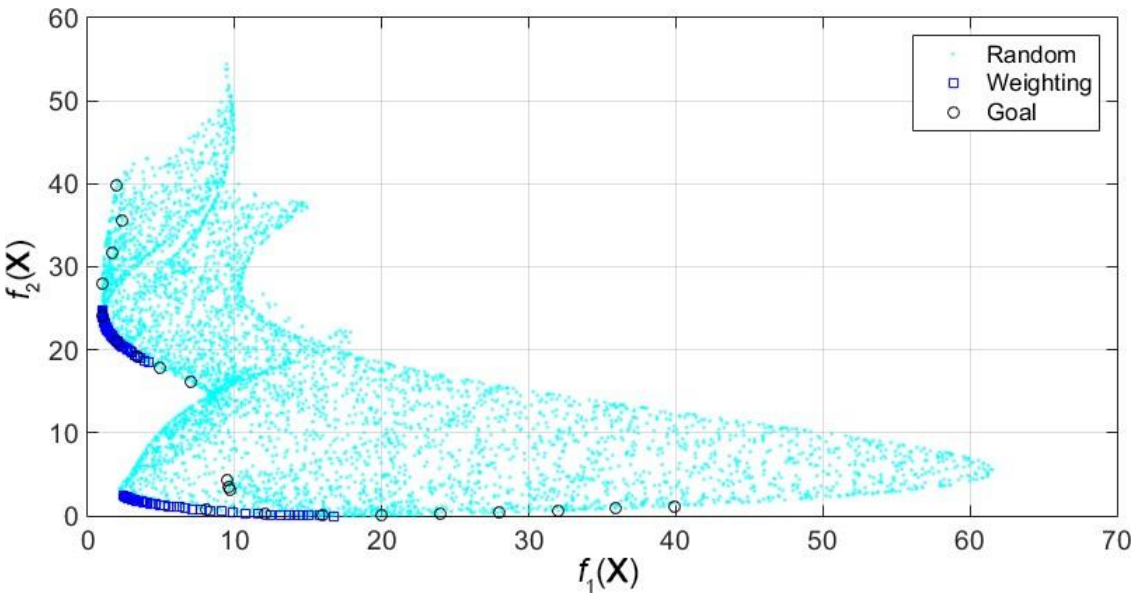

Figure 4 – Comparison of the solutions found for each strategy. Note that the goal-attainment and the weighted sum approaches had distinct approximated Pareto fronts according to their targets, as previously set by the decision maker. The random approach generated 10000 solutions which the decision maker would need to handpick the ones most suitable for him.

Through this technique a given set of solutions is provided with one being fairly separated from the other, enabling a greater diversity among the approximated Pareto front. On the other hand, this algorithm works better on MOPs with fewer objectives since it is hard to establish an Utopia hyperplane as well as its segments when the quantity ofobjectives increases.

For such complex cases where there is a higher amount of constraints and/or objectives involved the concept of Evolutionary Multi-objective Optimization (EMO) is an alternative since its algorithms (called within this research as multi-objective evolutionary algorithms or MOEAs) are more flexible, as noted by Coello (2006). The MOEAs are inspired by biological evolution mimicking actions such as reproduction, mutation and selection of the solutions in a way that successive generations are created as offspring of the previous one with the most suitable solutions handling over to the next generation their traits and attempting to improve over them as well. This way, independent on their original positions the solutions should move to the real Pareto front over time independent of the complexity of the feasible hyperspace and the constraints. Details on the MOEAcharacteristics will be explained on the Section 1.3.

Independently of the algorithm chosen during the MOO step, the desired output must always be a set of suitable solutions - since they are suitable, they must respect the MOP definition including all of its defined constraints.

The Multi-Criteria Decision-Making **(MCDM)** Bonissone, Subbu and Lizzi (2009) is the last process. After the approximate Pareto front had been determined as well as a number of solutions within it, these solutions should be shown to the decision maker in a helpful way. This in itself can also be a challenge because, for example:

- There are too many solutions to be shown in a Cartesian chart;
- There are too many objectives to be shown in a Cartesian chart - since each axis traditionally represent an objective, two objectives/axes can be easily represented. Three objectives/axes are more difficult depending on the generated Pareto front as the decision maker may take additional time rotating and changing the angle of the tridimensional chart in order to understand the tradeoffs. From four objectives and beyond it would be impossible to represent with ease the Pareto front.
- The differences between the generated solutions may be hard to represent within a single chart.

This means that the designer needs to spend more time to understand the tradeoffs between the proposed solutions. Depending on the quantity of generated data the designer may also find himself overwhelmed by it since it would involve the analysis of multicriteriasolutions scattered around multiple dimensions.

On the decision making, the designer usually have two choices: pass completely or partially the decision making responsibility to the algorithm through methods such as the minimum p-norm Mertins, Mei and Kallinger (2010) or the Nash procedure Coello (1999) where, for example, only one solution would be retrieved based on mathematical

processing over the whole list or through the multicriteria analysis. On the latter, the designer has a clearer view of the best solutions available and therefore may do faster decisions. This analysis - the MCDM - commonly may be done through two strategies -visually or through filtering and ranking.

On the filtering alternative, as shown by Antonio and Coello (2013), it is possible to use tools to do the post-processing of the list of solutions. This post-processing may include, for example, physical programming filters, restrictions and indicators, all informed by the designer. The result is, therefore, a filtered list with all the solutions meeting the criteria informed by the designer which may be compared as-is or even shown in a chart if needed. The physical programming, according to Messac (1996), is composed of a matrix with the preference values used for pertinence purposes. Within this matrix, a line would represent the objectives with the values representing the boundaries of highly desirable, desirable, tolerable, intolerable and highly intolerable values, respectively (i.e. the first and second values create the limit for highly desirable values, the second and third, for the desirable values, and so on). The values in this matrix are used to better choose the possible solutions depending on the designer preferences. Therefore, it may be used either for ranking or filtering purposes - the former by ranking first the solutions within the first range (i.e. highly desirable to desirable) and so on.

It is important noting that there are different strategies to the ranking option considering the complexity - and, therefore, performance - of the algorithm such as SMART, ANP, ELECTRE or SAW, as shown by Valiris, Chytas and Glykas (2005), Lee and Kim (2000), Roy (1990), Abdullah and Adawiyah (2014). One, proposed in 1993 by Hwang, Lai and Liu (1993), is called TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution). This strategy mainly works with two alternatives called *PIS and* NIS (positive and negative ideal solution, respectively). They are reminiscent of the concepts of the utopia and nadir points since the PIS is the theoretical point with the best attribute values while the NIS is the one with the worst attribute values. Naturally, in the MOO implementations the interest of the designer is to have the best possible solutions at hand. Having that said, the main objective is to have the chosen solutions both as near as possible to the PIS while being as far as possible to the NIS. As the TOPSIS' first step, the decision matrix is standardized for comparison purposes. This is done by obtaining the root of the sum of all the squared values for all of the solutions, for each one of the objectives. As a result, all of the criteria will have a resulting value obtained from all of the solutions. Then, the value for a given criterion for each one of the solutions is divided by its resulting value previously calculated. Therefore, all of the criteria results will be normalized.

As a second step, the values may be multiplied by weights given to each one of the criteria. From the resulting matrix, the third step is composed of building the PIS and

NIS are upon the best and worst values for each one of the criteria. Then, the fourth step will determine the separation of each one of the solutions from the PIS. This is done by calculating the squared difference between the normalized, weighted value of the criteria of a given solution to the PIS. These differences are then summed. The squared root of this sum is the *separation from the PIS*, or S_i^*. Similarly, the same process is also done to the NIS with the resulting value named S_i'. From both values, the relative closeness is determined for each solution by (1.10), where:

$$\frac{S_i'}{S_i^* + S_i'} \tag{1.10}$$

Therefore, each proposed solution will have a single value. These values may be easily sorted, grouped and/or ranked which means that the designer will have a resulting subset that may be analyzed and further compared in less time and with more simplicity.

Another strategy, as shown by Behzadian et al. (2010), is the PROMETHEE (Preference Ranking Organization Method for Enrichment of Evaluations). It is a ranking method which places positions to a given list of solutions based on two thresholds - preference and indifference (p_k and q_k) for each one of the criteria. One solution *i* is placed ahead of the other solution *j* (i.e. with a *preference factor* of 1) if the difference between its objective with the objective of the other solution (named $d_{i,j}(k)$) is greater than the preference value set by the system designer since *the difference is sufficiently big to justify reorganizing the ranking of the solution list*. If this difference is equal to or lesser than the indifference value it, even if it one solution has a value marginally better than the other the ranking would not be changed since *the tradeoffs are sufficiently small in the decision making process that they are negligible* (i.e. with a *preference factor* of 0). On the other hand, if the difference lays between both factors, the equation shown in (1.11) assigns an intermediate value between 0 and 1 in the preference factor.

$$P_{i,j}(k) \begin{cases} 0, & \text{if } d_k \le q_k \\ \frac{d_k - q_k}{p_k - q_k}, & \text{if } q_k < d_k \le p_k \\ 1, & \text{if } d_k > p_k \end{cases} \tag{1.11}$$

where, as previously mentioned, $P_{i,j}(k)$ represents the preference factor for the *k*-th objective for the *i*-th solution in relation to the *j*-th solution, $d_{i,j}(k)$ is the difference between the objective values of the *i*-th and *j*-th solutions and p_k and q_k are the preference and indifference values set for each objective.

Under multi-objective optimization cases, this is repeated for each one of the objectives - there is a preference degree which is a sum of the preference factors calculated to each one of the objectives multiplied by a weighting factor between 0 and 1.

In order to properly rank all of the available solutions, each solution has two scores - the positive and negative preference flows which state how a given action is preferred to all or by all the other solutions. These flows are made from the normalization of the preference factors. From both flows - with the positive one subtracting from the negative
- a single value is retrieved called *net preference flow*. Then, the algorithm described in Behzadian et al. (2010) is able to properly sort through all the solutions based from their net preference flows.

These strategies may be applied to a myriad of problems - three sample MOP cases, mentioned merely as examples and without any relation and/or strong resemblanceto the test subjects of this research, are:

1. I-BEAM design problem, presented by Huang, Gu and Du (2006) — a problem with two objective functions and four variables intended to minimize the total cross-sectional area and its deflection at the midspan of an I-beam considering two applied forces;

2. Nutrition problem, shown by Hwang, Lai and Liu (1993) — a problem with six variables and three objectives in order to balance the daily nutritional requirements of a person;

3. Automatic insulin administration problem, presented by Reynoso-Meza et al. (2013) — a problem containing three objectives and five variables in order to maintain the glucose levels in the bloodstream.

Position	T	P	PP
1	41	4	45
2	43	5	48
3	42	3	43
4	45	1	51
5	40	2	52
6	46	6	41
7	44	9	42
8	38	12	40
9	47	8	46
10	48	10	47

Table 1 – Top 10 solutions for the I-BEAM problem for one sample run. T = TOPSIS (Tech- nique for Order Preference by Similarity to Ideal Solution); P = PROMETHEE (Preference Ranking Organization METHod for Enrichment of Evaluations); PP = Physical Programming.

All of those problems are based on the general MOP problem which is stated inthe equation (1.4).

The positions for each solution depends on the strategy used, as seen in the Tables1, 2 and 3, for example.

Position	T	P	PP
1	841	297	643
2	643	608	723
3	723	747	841
4	123	191	892
5	373	308	123
6	646	957	773
7	652	348	373
8	699	95	138
9	54	71	646
10	518	711	652

Table 2 – Top 10 solutions for the Nutrition problem for one sample run. T = TOPSIS; P = PROMETHEE; PP = Physical Programming.

Parameter	$C\underline{1}$	$C\underline{2}$	$C\underline{3}$	$C\underline{4}$	$C\underline{5}$
Insignificant	3.00	0.10	0.10	0.02	0.50
Significant	10.00	0.30	0.50	0.04	1.00

Table 3 – PROMETHEE rankings, Insulin problem. The columns show the values for each criterion ($C\underline{1}$, $C\underline{2}$, $C\underline{3}$, $C\underline{4}$, $C\underline{5}$). The numbers besides significant and insignificant show the number of the run.

The second MCDM strategy - visual - revolves around the concept of graphically showing the possibly solutions through a shared chart. This chart is *n*-dimensional where *n* is the number of objectives. All of the solutions are shown plotted on it where their positions are set by the mathematical values assigned to each of their objectives.

While the visualization through charts is a great tool in order to quickly assess the possible solutions when there the problem has two dimensions (i.e. objectives), the visualization suffers from readability issues when there are three or more dimensions involved. Since MOO usually have more than two objectives, there are some techniques to better display in a graphically fashion the possible solutions found. Some of them are, for example:

1. Scatter plots;
2. Parallel coordinates, as shown by Inselberg (2014);
3. Horizontal bars, noted in Miettinen (2014);
4. Radar plots, explained in Ward, Grinstein and Keim (2015).

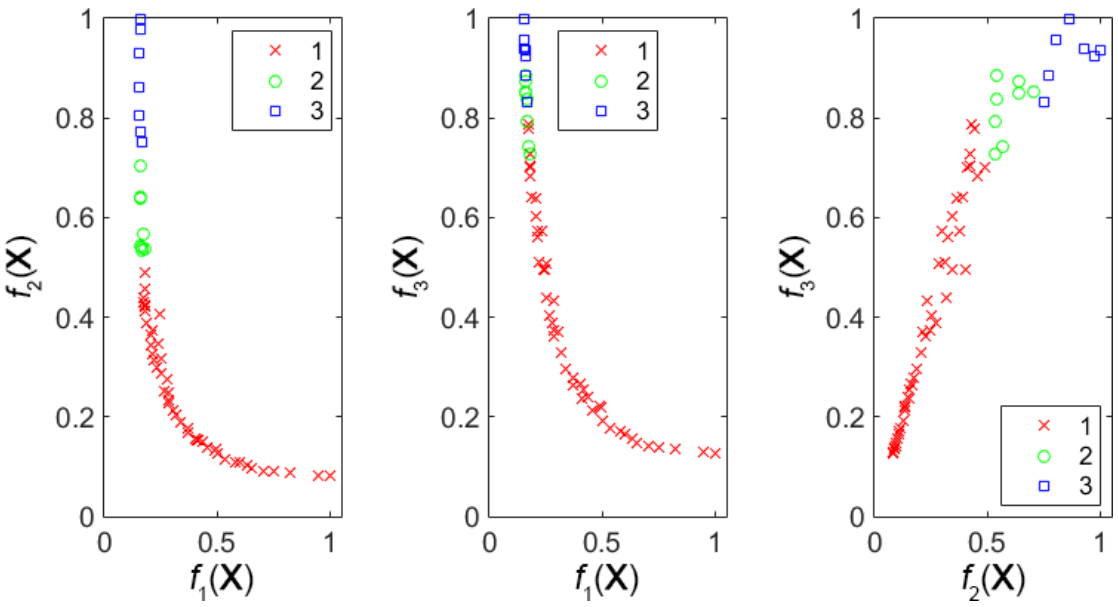

Figure 5 – Example of the scatter plot for the I-BEAM problem.

Depending on the number of solutions it also might make sense to combine the ranking/filtering techniques along with the charts. As seen in the Figure 5, for example, the solutions were previously divided into three clusters. In the chart it is therefore easier to understand the placement of the solutions found for each cluster.

The scatter plot, as seen in the Figure 5, works by displaying a n-dimensional space in a set of 2-dimensional plots. In an example of three dimensions, three charts would be shown - one with the first vs. second dimension, other with the first vs. third dimension and another with the second vs. third dimension, with all of them displaying the samedata set, only within different perspectives.

The parallel coordinates approach, as seen in the Figure 6, uses a single 2-dimensional space in order to display all the solutions. With each objective being presented within the *x*-axis and their values in the *y*-axis, each objective is represented as a line.

The horizontal bars, according to the Figure 7, displays one single plot where each bar refers to the values found for a given objective of a given problem. In this case, the bars are grouped by objective - however, it is equally feasible to group by solution instead as shown in Miettinen (2014). In both cases, the bars will represent the values found for each solution and objective combination.

Another approach, as seen in the Figure 8, is called radar plots. Analogous to the parallel coordinates, a single plot shows all the objectives and all of the solutions as lines. The main difference is that, as explained by its own name, the plot type used by this strategy is the radar.

In order to properly display the data using these approaches there are software tools

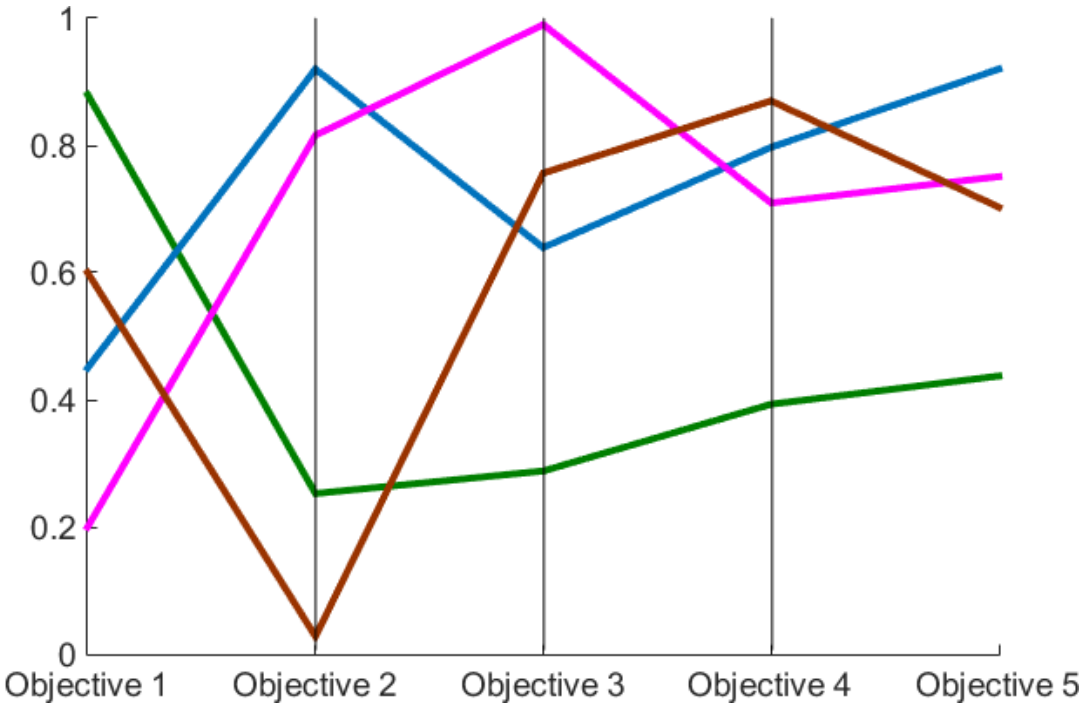

Figure 6 – Example of parallel coordinates for an unrelated problem. The *y*-axis represent the normalized values found for each objective.

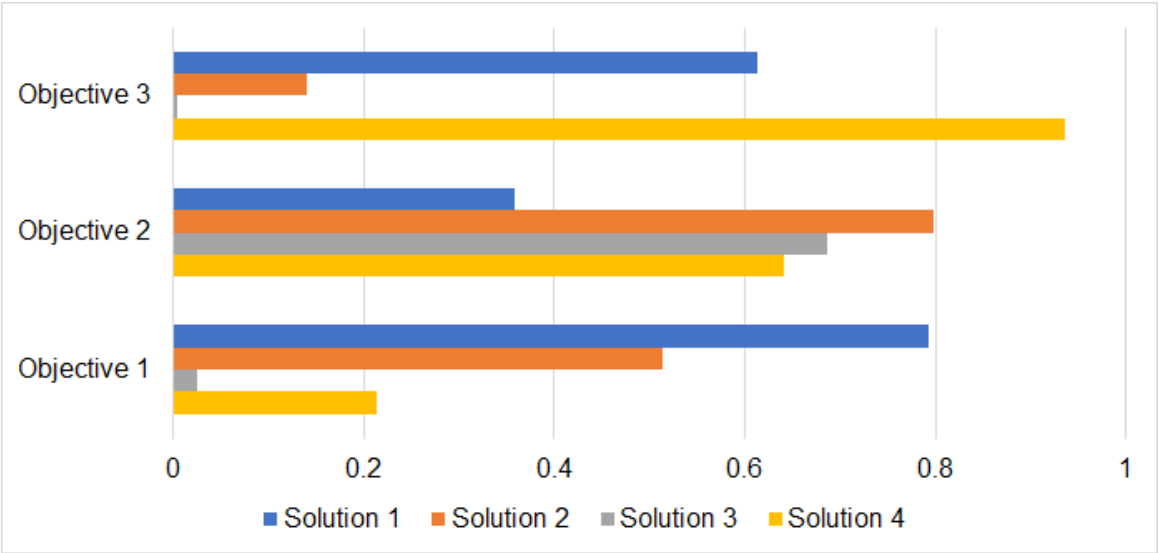

Figure 7 – Example of the horizontal bars visualization of a random problem containing three objectives.

available such as Sliver, shown in SOFTWARE (2016); XDAT, shown by XDAT (2016) and Grapheme, as shown in ICHROME (2016) in order to properly display the parallel coordinates data. For the scatter plots, there are commonly plugins for Excel such as SigmaXL, according to SIGMAXL (2016) and software such as JMP, shown in JMP (2016). On the level diagrams, there are papers with suggested approaches such as

the ones shown

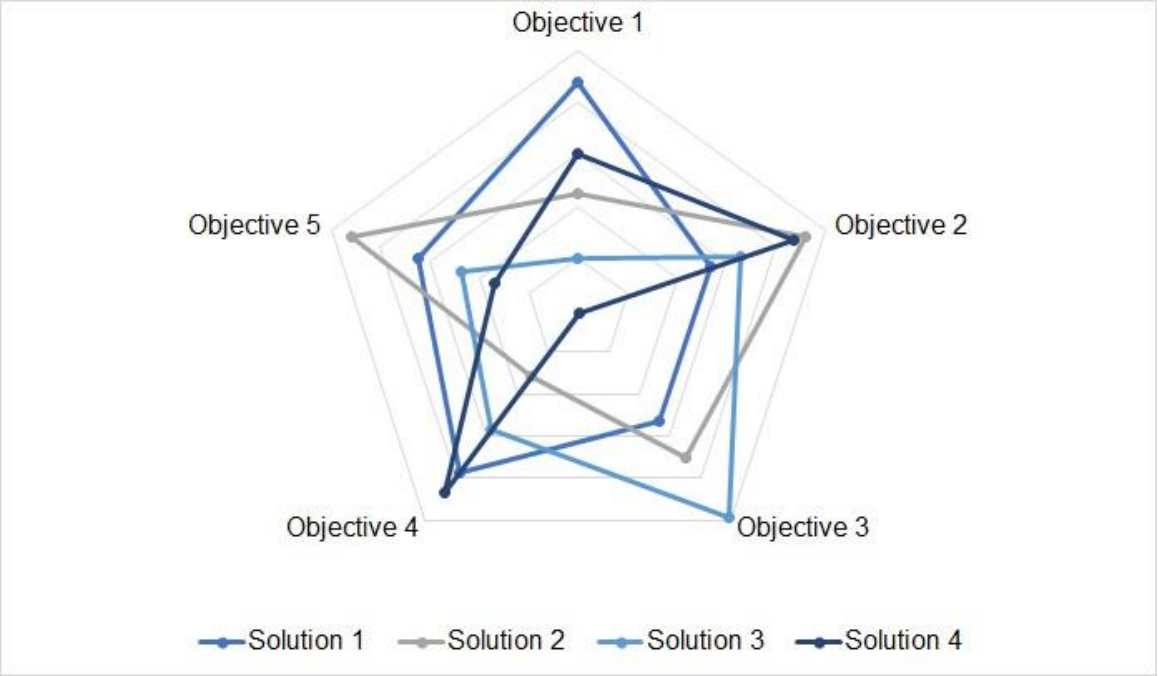

Figure 8 – Example of the radar plot for an unrelated problem.

in Blasco et al. (2008) and Reynoso-Meza (2014). For the radar plots, there are approaches such as Origin, mentioned in ORIGINLAB (2016) and Plotly, shown in PLOTLY (2016). Apart from all of them, most computing environments (such as MATLAB®) also offers some or all of the visualization techniques aforementioned.

1.3 Multi-objective Evolutionary Algorithms

When discussing the Multi-objective Evolutionary Algorithms (MOEAs) there are a myriad of techniques and variants well explored: examples are the Particle Swarm Optimization (PSO), the cuckoo search, Genetic Algorithms (GAs), Artificial Bee Colony (ABC), Ant Colony Optimization (ACO). Like the SO counterparts, each of these strategies have tradeoffs when compared against each other. In fact, they were originally developed for SO, being later adapted for EMO purposes. A general MOEA is described in the Algorithm 2.

Independently of the algorithm chosen, there are some desired characteristics to be looked for, as mentioned and shown by Coello et al. (2007), Corne and Knowles (2007), Reynoso-Meza (2014), Beyer and Sendhoff (2007), Santana-Quintero, Montano and Coello (2010), Lozano, Molina and Herrera (2010), Farina, Deb and Amato (2004), Das et al.(2011):

- **Convergence:** this characteristic express the capacity of an algorithm to reach the real Pareto front that, as mentioned earlier in the section 1.2.1, is unknown.

Algorithm 2: Basic MOEA, as shown in Reynoso-Meza (2014)

Data: objective vector, design space

Result: the Pareto set approximation $\mathcal{X}_p^*|_G$

1 build initial population $P|_0$ with N_p individuals;
2 evaluate $P|_0$;
3 build initial Pareto set approximation $\mathcal{X}_p^*|_0$;
4 set generation counter $G = 0$;
5 **while** *convergence criteria not reached* **do**
6 $G = G + 1$;
7 build population $P^*|_G$ using $P|_G$ with an evolutionary technique;
8 evaluate new population $P^*|_G$;
9 build Pareto set approximation with $\mathcal{X}_p^*|_G$ with $\mathcal{X}_p^*|_{G-1} \cup P^*|_G$;
10 update population $P|_{G+1}$ with $P^*|_G \cup P|_G$;
11 **end**

- **Diversity:** this characteristic externalizes the capacity of a given algorithm to generate a set of *distributed* solutions. For the decision maker, it is better if he could have available a list of ten different, well-spread solutions along the Pareto front than ten solutions almost equal between each other while other sections of the Paretofront would be left without any solution.

- **Pertinence:** for the decision maker is also important to have solutions that might be more useful for his needs. That is, from the generated Pareto front it might make sense for him, for example, to use only the solutions that are able to reach an equilibrium along all the objectives - i.e. ignore all the solutions that are feasible or impractical for at least one objective. If the decision maker knows the kind of solutions he is looking for he is able to use a MOO algorithm capable to set up his pertinence preferences before the algorithm execution. Or, for other algorithms, the decision maker can choose a desirable set of solutions *after* the algorithm is executedor even while the optimization is in progress.

- **Constrained optimization:** like the other characteristics, the constrained optimization Zhou et al. (2011), Bonyadi et al. (2016) is also a desirable characteristic because most of the MOPs also include constraints. This, in other words, determine the capacity of an algorithm to determine whether if a solution is feasible or infeasible. This generally is achieved through some techniques which include, but are not limitedto:

  Feasibility rules: when selecting the best solutions for reproduction to the next generation, generally the solutions are compared against each other. If the subjects of a given comparison are two feasible solutions, the one with the best objective function will be selected. In the other hand, if two infeasible solutions are compared, the one that is less violating the constraints will be selected. If, however, a feasible

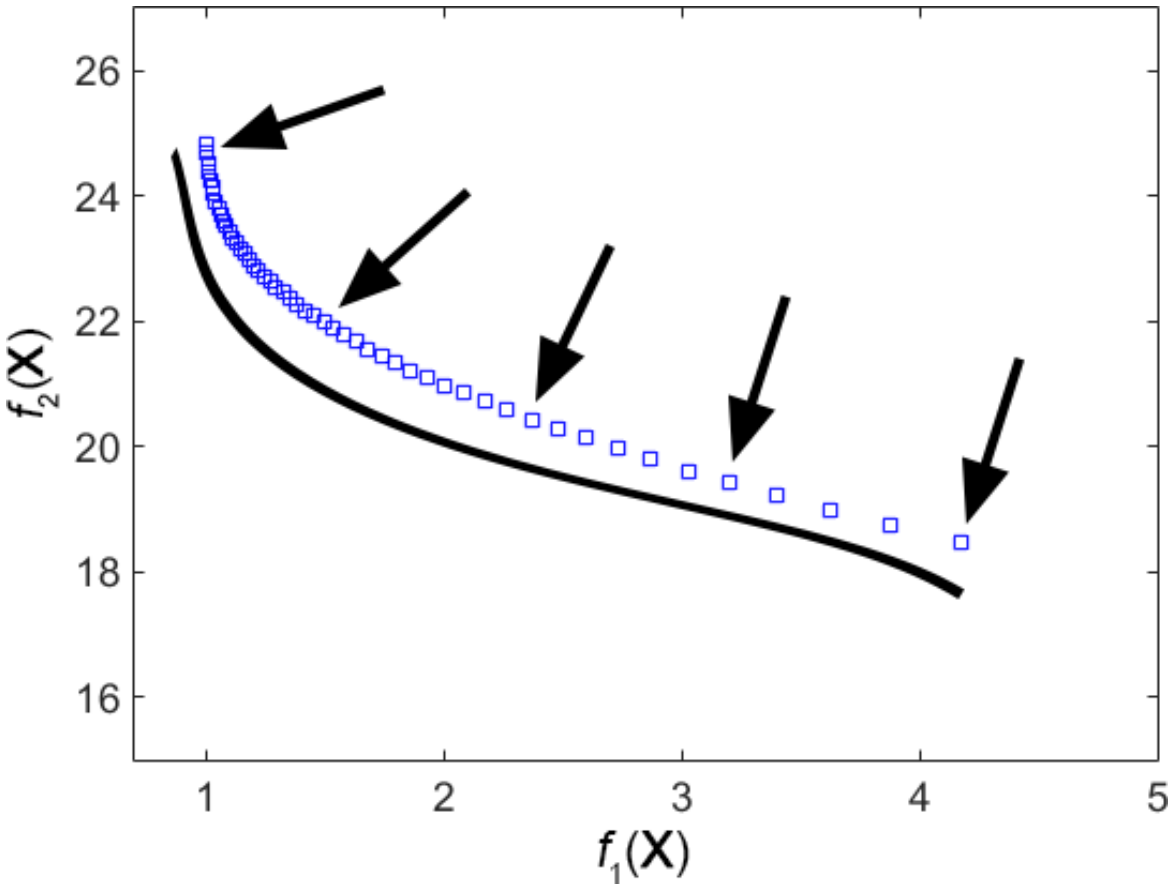

Figure 9 – Example of the convergence concept for a multi-objective evolutionary algorithm, where the continuous curved line represents the real, unknown Pareto front, and the squares were the results found by an algorithm. Under the convergence characteristic an algorithm must found solutions as close as possible to the real Pareto front - i.e. after each iteration the solutions would follow the direction shown by the arrows until they theoretically reach the real Pareto front.

solution is being compared against an infeasible solution, the feasible solution should be used.

Stochastic ranking and novel penalty functions: penalty scores and special opera- tors are implemented to handle over solutions that do not meet the requirements of the constraints. Under such scenarios the objective function value is modified in order to differ them from other valid solutions - for minimization problems a high value might overwrite the calculated objective values, for example. Depending on the number of constraints breached this number might increase meaning that both the feasible solutions can be compared and ranked against each other as well as the infeasible ones. Please note, however, that a feasible solution does not necessarily mean that it is also part of the approximated Pareto front as explained in the section 1.1. Novel penalty functions and their operators can implement different approaches, although based on the same concept of

differing and organizing the

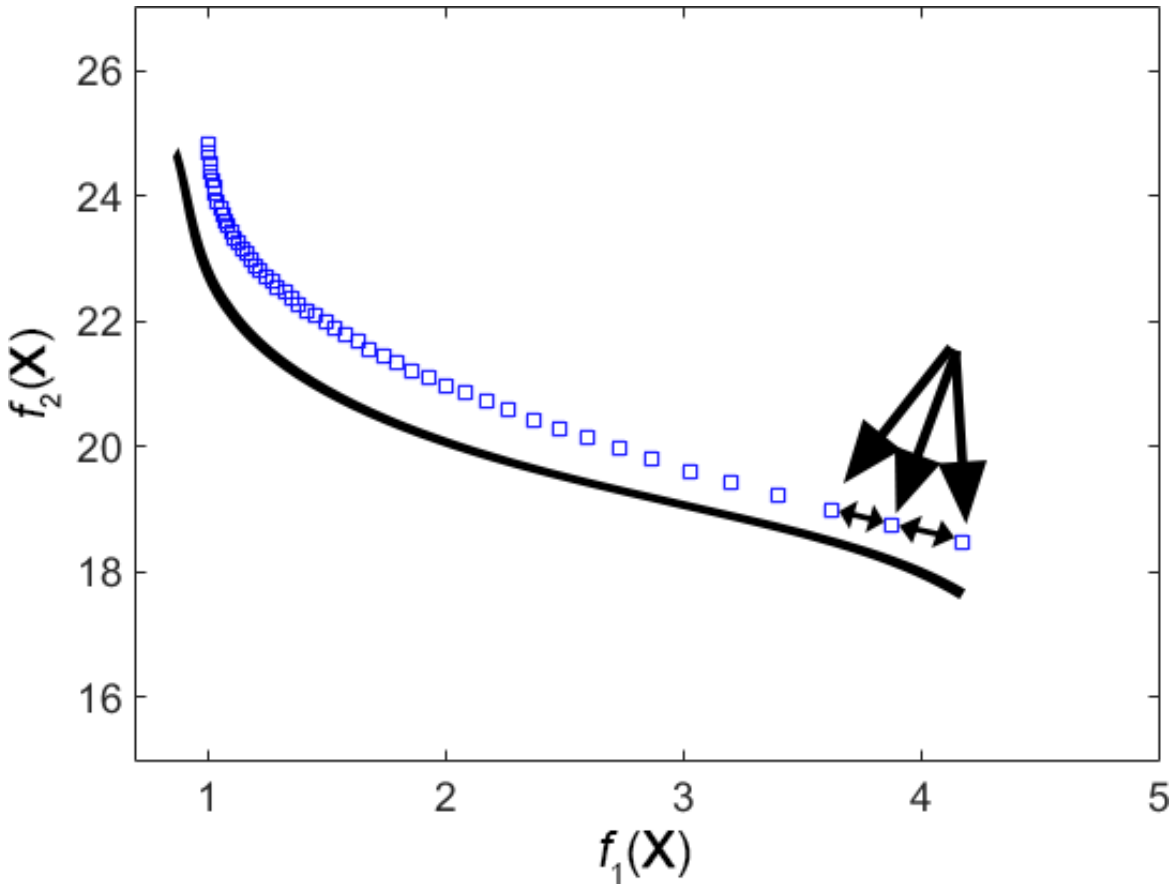

Figure 10 – Example of the diversity characteristic for a multi-objective evolutionary algorithm, where the continuous curved line represents the real, unknown Pareto front, and the squares were the results found by an algorithm. Under this characteristic the solutions must be both spread along the real Pareto front as well as distributed among themselves, as shown by the solutions highlighted with the arrows - notice the distance across them.

infeasible solutions from the solution ones.

Multi-objective concepts: in this scenario, the constraint can be treated as anadditional objective to be optimized.

- **Many-objectives Optimization:** one of the challenges the algorithms should resolve is the optimization in a *m*-dimensional objective space if the solutions are dominance resistant while attempting to keep an acceptable degree of diversity, specially when there are five or more objectives involved Cai et al. (2015). Techniquesto resolve this situation could be:

  Reducing the number of objectives;

  Making use of the region of interest for the decision maker;

  Define a metric based on the number of objectives to determine *what* would define when a feasible solution is better than the other;

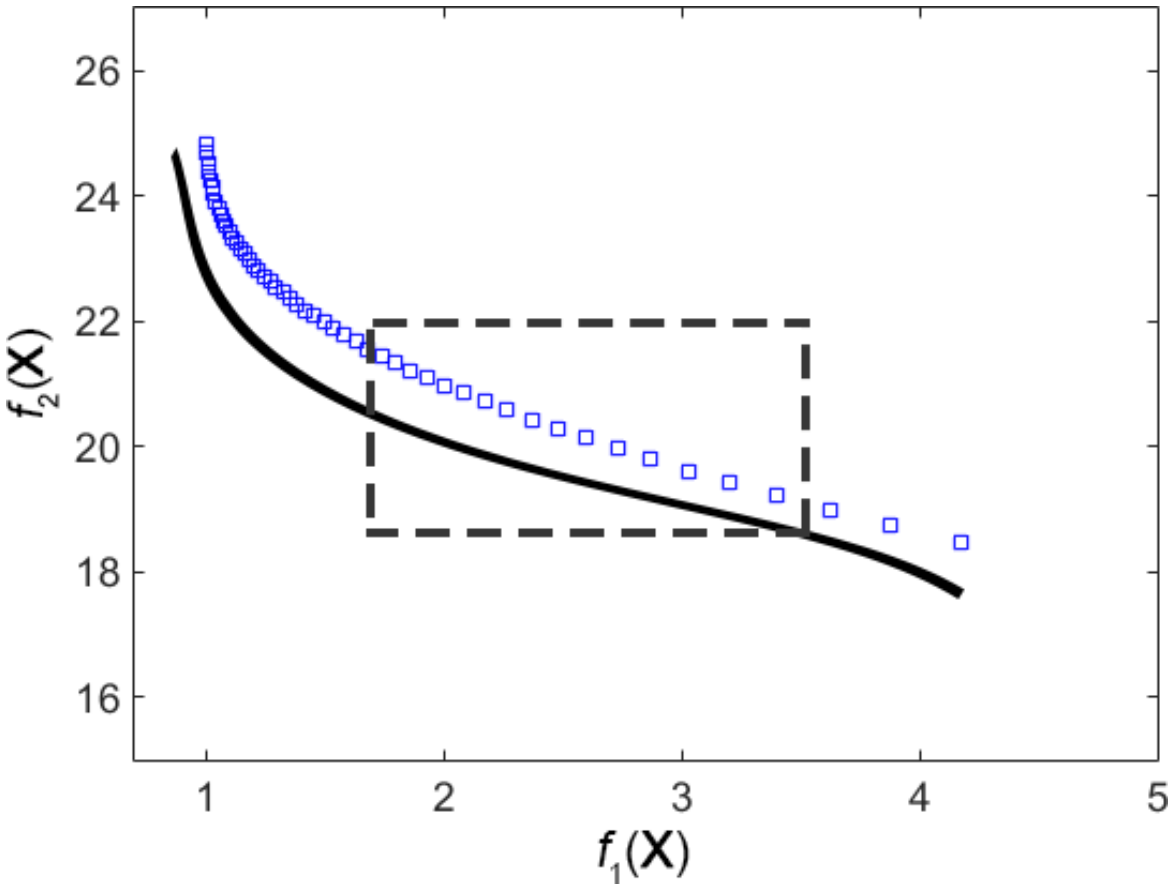

Figure 11 – Example of the pertinence characteristic for a multi-objective evolutionary algorithm, where the dashed box represents an example of a region of interest to the decision maker. Depending on the algorithm it is possible to determine the area beforehand or determine it after the algorithm execution. In this case, the decision maker would need to manually filter out the solutions most useful for him - i.e. only the ones inside the box.

Modification of the Pareto dominance logic, therefore changing the selection pressure towards the Pareto front;

Using scalarizing functions;

Using performance indicators of the quality of the Pareto front approximation.

- **Dynamic Optimization:** under some scenarios, specially when involving motors and control-based applications, there is an additional requirement to modify the objective function dynamically depending on the current situation Farina, Deb and Amato (2004) - i.e. the objective function in use halfway through the MOO process might differ from the one used in the initial iteration.

- **Multi-modal Optimization:** similarly, for other real world applications there might be in the approximated Pareto front some solutions sharing the same objective vector Qu et al. (2016) - i.e. they might meet all the constraints and have the

same result. However, for the decision maker it can make sense to see both of them instead of just one, selected previously by the algorithm itself. This case can appear when other subjective and/or tangible aspects can be considered: back in the production plan example, two solutions can achieve the same profit and costs of production. However, they have minor Legal and HR differences that would require the comparison and decision from these departments. In this scenario, it would makesense to have available both solutions.

- **Robust Optimization:** this characteristic refers to the models used to measure the robustness of the algorithm Gabrel, Murat and Thiele (2014) - i.e. how much, by the influence and presence of uncertainities, the objective vector could be degraded.

- **Computionally Expensive Optimization:** Depending on the complexity of the objective function, its evaluation can use too many computer resources Akhtar and Shoemaker (2016). Depending on the case, this expense could be reduced by Neural Networks or through the generation of a surface within the objective spaceon runtime.

- **Large-scale Optimization:** Since different algorithms have tradeoffs when compared against each other, logically some of them should perform better under some situations than others. By extension, some algorithms should perform quite well within solutions with a relatively small amount of variables while having a poor performance or even not working at all. Therefore, a desirable capacity of any MOO algorithm is to solve any problem regardless of the quantity of variables involved - even when the problem is considered as large-scale such that it has a high amount of decision variables Cheng et al. (2016). More on this topic will be discussed on the Section 1.4.

1.4 Large-scale Optimization

According to Cheng et al. (2016), a *large-scale problem* generally has hundreds or thousands of decision variables. By increasing the number of variables, the volume of the search space and its complexity also increases exponentially in what is called *curse of dimensionality*, according to Bergh and Engelbrecht (2004). Also, many evolutionary algorithms that perform well with low-dimension problems often fail to approximate to the real Pareto front, as shown by Yang, Tang and Yao (2008).

For this reason, one of the proposed approaches to large problems is to separate the search space into smaller, lower dimensional subspaces in what is called Cooperative Coevolutionary Genetic Algorithm (CCGA), explained in Potter and Jong (1994). This algorithm has the following assumptions:

- A *species* is a subcomponent of a potential solution;
- By combinating members of each one of the species available it is possible to findcomplete solutions;
- The fitness of a given subpopulation member is determined through combination with the current best subcomponents of the other subpopulations;
- The evolution of each subpopulation is given by a standard genetic algorithm (GA);
- When required, the number of subpopulations will evolve by itself.

The original algorithm, therefore, works under the following structure:

Algorithm 3: CCGA

Data: objective vector
Result: the Pareto front approximation
1 separate the objective vector into one-dimensional subcomponents;
2 **for** *each segment* **do**
3 optimize cooperatively the subcomponent through an evolutionary algorithm through a fixed number of fitness evaluations or until the stopping criterion is satisfied.
4 **end**

However, although this solution can lead to significant performance improvements, it may deteriorate when there is a dependence among parameters. Furthermore, the partitioning may create pseudominima, as explained in greater detail by Bergh and Engelbrecht (2004) - that is, minima created as a side effect of the aforementioned partitioning of the search space.

Also, since this concept mainly revolves around the problem decomposition, it is by itself critical. The existing techniques mainly use two simple decomposition methods, as shown in Yang, Tang and Yao (2008). The first one, one-dimensional based, simply decomposes a high-dimensional vector into individual, separate variables. This was the strategy used by the CCGA originally proposed. The other option, splitting-in-half, decomposes the vector into two halves. Due to their nature, their shortcomings are foreseeable - the first strategy do not consider interdepencies between the variables while the second one may still suffer from performance issues depending on their size. Furthermore, if two given variables are dependent among themselves but are located along the original vector in a way that they do not belong to the same half after the split their interdependency may be affected. As it was also mentioned in Yang, Tang and Yao (2008), cooperative co-evolutionary algorithms were rarely tested against high dimensionalproblems (i.e. with more than 100 dimensions).

In order to tackle down the shortcomings from the original CCGA and/or to adapt it for multi-objective problems, alternatives such as the cooperative particle swarm optimization (CPSO), shown in Bergh and Engelbrecht (2004); the Self-adaptive Neighbourhood Search Differential Evolution algorithm (SaNSDE), shown in Yang, Tang and Yao (2008); the Multi-objective Cooperative Coevolutionary Genetic Algorithm (MOCCGA), explained by Keerativuttitumrong et al. (2002) and the Cooperative Coevolutionary Generalized Differential Evolution (CCGDE3) algorithm were proposed over the last years, as seen in Antonio and Coello (2013). The cooperation logic of the latter works through the following structure:

Algorithm 4: Cooperation logic for the CCGDE3 algorithm

Data: a set of components (generation)
Result: the Pareto front approximation
if *it is the first generation* **then**
  form random collaborations between the subpopulations;
else
  from the best non-dominated species in the subpopulations of the previous population form new collaborations with randomly selected components;
end
evaluate these collaborations - i.e. individuals forming a set of solutions - in their objective functions;
assign back the results to the individuals under evaluation.

In general, it differs from the original CCGA by separating the objective vector into S subcomponents of equal size where each one represents a subset of all the decision variables at a time. Each subcomponent is created by randomly grouping the variables. Also, S subpopulations are created with a given number of individuals, where the decision variables are randomly assigned to each subpopulation - i.e. a subcomponent corresponds to a subpopulation. This algorithm is based on the third version of the Generalized Differential Evolution (GDE3) algorithm.

1.5 Hybrid Optimization

It is important to note that GAs in general require a great number of iterations and also they converge slowly - the nearer they are from the global optimum the slower they get. As explained in Kelner et al. (2008), due to this weakness, one proposal is to enhance a GA with an additional local search step before the selection of the individuals, for example. This way, the individuals should be optimized to their locally optimal positions before going to the next generation - in other words, the generations are enhanced and their evolution speed is increased. Therefore, less iterations should be needed by the algorithms.

There are a myriad of algorithms including recent implementations such as the ones shown in Lin et al. (2015), Harris, Mumford and Naim (2014), Kuroda et al. (2015). Another example, as shown by Sindhya, Deb and Miettinen (2011), involved a modified version of the NSGA-II algorithm, a well-known genetic algorithm also explained by Deb et al. (2002). A new operator responsible for the local search was introduced in the algorithm as well as a new convergence-based termination criterion.

In the Algorithm 5, the stopping criteria can include, but are not limited to: timeout, number of generations without improvement or maximum number of generations. In short, the algorithm works as follows:

Algorithm 5: Hybrid optimization algorithm

Data: design space, objective vector
Result: the Pareto front approximation

1 separate the objective vector into one-dimensional subcomponents;
2 generate a random initial population;
3 **while** *the stopping criteria are not satisfied* **do**
4 sort it to different non-domination levels, assigning to each one of the individuals a fitness equal to the level they belong to;
5 create offspring through binary tournament, recombination and mutation;
6 perform local search on some individuals picked through a probability value from the offspring. The picked individuals will be replaced by the improved individuals resulted from the local search;
7 combine both parent and offspring populations;
8 execute non-dominated sorting and identify the different non-domination levels again;
9 set a new empty population and populate it from the best down to the worst level while the population size is less than or equal the original population size;
10 run a crowding-sort procedure, including the most widely spread individuals of each level through the crowding distance values;
11 form new collaborations with randomly selected components from the best non-dominated species in the subpopulations of the previous population;
12 evaluate these collaborations - i.e. individuals forming a set of solutions - in their objective functions;
13 assign back the results to the individuals under evaluation;
14 **end**

In fact, while hybrid optimization strategies are more used in combinatorial optimization problems, as seen in Ahn et al. (2010) they can also be implemented in problems within a continuous domain, similar to the ones that will be used in this research.

1.6 Benchmarks

Considering the shortcomings of the different strategies presented in the previous sessions it is worth noting the requirement to compare the performance of them all under standardized tests. Since this research intends to show a new approach on multi-objective optimization problems more likely to happen in real world scenarios, the aim is to test it against such problems.

From a implementation perspective, one of the natural challenges is to test the algorithm against more complex problems due to their size and the time taken to address any issue found and/or to optimize the algorithm itself. Let us consider a given problem that took eight hours to run in a given machine - should any error appear at the end of the process or during the process, all of the hours taken during the processing will be wasted. Additionally, depending on the implementation techniques used there might be complicated issues to address and debug on execution time that might get harder accordingto the MOP used - e.g. if there are three or more objectives in use.

Furthermore, for performance tuning purposes it is interesting to use smaller, mathematical MOPs - since the results are more predictable, it is easier to find potential bottlenecks prone to optimization. After the results are satisfactory bigger scenarios can be evaluated and compared against other known algorithms. It might be possible to find out that the algorithm to be proposed performs better or worse depending on the type of the problem.

If the new algorithm is quicker on the first, smaller tests against other algorithms but is unacceptably slower as the problem scales up while the quality of the Pareto front approximation is acceptable it might mean that the algorithm may have potential bottlenecks - if there is not a obvious roadblock in the pseudocode then the implementation itself could be tuned - e.g. the parallelism settings could be adjusted, a possible sorting criterion could be fixed or a programming language particularity could be subject of a workaround. On the other hand, the new algorithm might quickly generate the final approximate Pareto front under any test of any size. However, the results can be considerably farther from the real Pareto front as the problem scales up in size. In this scenario, the algorithm must be tested again through bench testing with the pseudocode and, if it proves that it is not causing the issue, then the code implemented must be checked for possible issues when iterating through the different objectives.

For these reasons, a list of tests was created, as follows:

- A set containing 9 test problems reminiscent of industrial (production line) problems, shown in Cheng et al. (2016);
- Poultry dataset.

A set of large-scale multi-objective test problems had been created by Cheng et al. (2016). According to Cheng et al. (2016), previous test sets did not consider different characteristics of the decision variables, even if in theory they are scalable to support any number of decision variables. This test set made use of the following basic principles, as recalled by Deb et al. (2005), Huband et al. (2006):

- Scalable to have a given number of objective functions;
- Scalable to have a given number of decision variables;
- Known, easy to understand shapes and locations of the Pareto front approximations;
- Generated with a uniform design formulation.

Overall, in this set the formulation used was:

$$\begin{aligned} & f_1(x) = h_1(x^f)(1 + g_1(x^s) \\ & f_2(x) = h_2(x^f)(1 + g_2(x^s) \\ & \dots \\ & f_M(x) = h_M(x^f)(1 + g_M(x^s) \end{aligned} \tag{1.12}$$

With M objective functions, where f_n is an objective function. The decision vector x was split in two parts - x^f and x^s. The h_n functions are responsible for defining the Pareto front shape while the g_n functions are in charge of defining the fitness landscape.

The set also have these characteristics:

- Nonuniform grouping of decision variables;
- Different correlations between variable groups and objective functions;
- Mixed separability between the decision variables;
- The decision variables have linkages on the Pareto sets.

All of the proposed tests shares these characteristics by implementing, for example, a chaos-based pseudo random number generator (to address the nonuniform grouping), a correlation matrix to keep track of the correlations between variable groups and the objec- tive groups, different basic fitness landscape functions (to achieve the mixed separability) and also making use of a linkage function to define the variable linkages.

All the tests uses a combination of six basic single-objective functions. These

functions are the sphere function, the Schwefel's function, the Rosenbrock's function, the Rastrigin's function, the Griewank's function and the Ackley's function.

Sphere function, an unimodal and separable problem:

$$\eta_1(x) = \sum_{i=1}^{|x|} (x_i)^2 \tag{1.13}$$

Schwefel's function, an unimodal and non-separable problem:

$$\eta_2(x) = \max_i \{|x_i|, 1 \leq i \leq |x|\} \tag{1.14}$$

Rosenbrock's function, a multi-modal and non-separable problem:

$$\eta_3(x) = \sum_{i=1}^{|x|-1} [100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2] \tag{1.15}$$

Rastrigin's function, a multi-modal and separable problem:

$$\eta_4(x) = \sum_{i=1}^{|x|} (x_i^2 - 10cos(2\pi x_i) + 10) \tag{1.16}$$

Griewank's function, a multi-modal and non-separable problem:

$$\eta_5(x) = \sum_{i=1}^{|x|} \frac{x_i^2}{4000} - \prod_{i=1}^{|x|} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{1.17}$$

Ackley's function, a multi-modal and separable problem:

$$\eta_6(x) = -20exp\left(-0.2\sqrt{\frac{1}{|x|}\sum_{i=1}^{|x|} x_i^2}\right) - exp\left(\frac{1}{|x|}\sum_{i=1}^{|x|} cos(2\pi x_i)\right) + 20 + e \tag{1.18}$$

All the problems part of this set have user-defined parameters for the number of variables and number of objectives. As such, these quantities are later defined in theSection 4.1. The problems composing this test set are the following:

1.6.1 LSMOP1

This is an unimodal problem with a fully separable fitness landscape. It generates a linear Pareto front and has following variable linkage function:

$$\begin{aligned} x^8 &\leftarrow \left(1 + \frac{i}{|x^8|}\right) \times (x^8 - l_i) - x^f \times (u_i - l_i) \\ i &= 1, \ldots, |x^8| \end{aligned} \tag{1.19}$$

This function has the following objective functions:

$$f_1(x) = x^f \ldots \quad 1 + \sum_{j=1}^{M} c_{1,j} \times g \quad (x^8)$$

$$f_2(x) = x^f \ldots (1 - x^f) \quad 1 + \sum_{j=1}^{M} c_{2,j} \times g \quad (x^8)$$

$$\vdots$$

$$f_{M-1}(x) = x^f (1 - x^f) \quad 1 + \sum_{j=1}^{M} c_{M-1,j} \times g \quad (x^8)$$

$$f_M(x) = (1 - x^f) \quad 1 + \sum_{j=1}^{M} c_{M,j} \times g \quad (x^8) \tag{1.20}$$

with

$$c_{i,j} = \begin{cases} 1, \text{ if } i = j \\ 0, \text{ otherwise} \end{cases} \tag{1.21}$$

and

$$g(x^8) = \frac{1}{n_k} \sum_{i,j} \eta_1(x^8)$$

$$g \quad (x^8) = \frac{1}{n_k} \sum_{i,j} \eta_1(x_{i,j}^8)$$

$$n_k, \quad j = \quad i,j \quad |x^8 - x^8| \tag{1.22}$$

$$2k \quad i$$

$$k = 1, \ldots, \quad M$$

1.6.2 LSMOP2

This is a problem with mixed modality and partially separable fitness landscape.

–

It generates a linear Pareto front and has following variable linkage function:

. 1 M−1 j=1 2 j

... (1.24)

.

$$f_M(x) = (1 - x_1^f) \sum_{1} + \sum_{j=1}^{M} c_{M,j} \times \bar{g}_M(x_j^8)$$

$$x^8 \leftarrow 1 + \frac{i}{} \times (x^8 - l_i) - x^f \times (u_i - l_i)$$

|x8| i 1

□i = 1, ., |x8|

(1.23)

$$1 + \sum_{M} c_{1,j} \times g\ (x^8)$$

$f_1(x) = x^f \ldots$

x^f

1 $M-1$ □

$j=1 \qquad 1\ j$

$$\square f_2(x) = x^f \ldots (1 - x_g^f) \ 1 + \sum^M \ c_{2,j}$$

$$f_{M-1}(x) = x^f(1 - x^f) \quad 1 + \sum_{M}$$

$$1 \qquad f=1$$

with

$$c_{i,j} = \begin{cases} 1, \text{ if } i = j \\ 0, \text{ otherwise} \end{cases} \tag{1.25}$$

and

$$g^1(x^8) = \frac{1}{n_k} \sum_{i,j} \eta_5(x^8_{i,j})$$

$$g^2(x^8) = \frac{1}{n_k} \sum_{i,j} \eta_2(x^8_{i,j}) \tag{1.26}$$

$$\eta_5(x^8_{i,j}) = \begin{cases} 1, \text{ if } x^8_{i,j} \neq 0 \\ 0, \text{ otherwise} \end{cases}$$

$$\eta_2(x^8_{i,j}) = \begin{cases} 1, \text{ if } x^8_{i,j} \neq 0 \\ 0, \text{ otherwise} \end{cases}$$

$$n_k = \left| x^8_{i,j} \right|, \quad i, j = 1, \ldots, \frac{n}{2}$$

$$k = 1, \ldots, M-1$$

1.6.3 LSMOP3

This is a multi-modal problem with a mixed separable fitness landscape. It generates

$$\ldots \tag{1.28}$$

a linear Pareto front and has following variable linkage function:

$f_1(x) = x^f \ldots$ x^f

$$1 + \sum_{M} c_{1,j} \times g\ (x^8)$$

$$f_2(x) = x^f \ldots (1 - x^f \quad) \quad 1 + \sum_M \quad c_{2,j} \times g$$

$$(x^8)$$

$$f_{M-1}(x) = x^f (1 - x^f) \quad 1 + \sum_M$$

$$c_{M-1,j} \quad \times g$$

(x^8)

$1 \quad \not=1$

$M-1\,j$

$$\frac{G_{M,i}}{GM} \times$$

$$c_{i,j} = \begin{cases} 1, \text{ if } i = j \\ 0, \text{ otherwise} \end{cases}$$

$$\sum_{k=1}^{n_k}$$

$$\square g$$

$$(x^8) = \qquad \eta_4(x^8 \quad)$$

$$\sum_{j=1}^{i} n_K$$

$$n \quad j=1$$
$$, \quad ,$$

***i*, *j* |**

1.6.4 LSMOP4

This is a problem with mixed modality and a mixed separable fitness landscape. It generates a linear Pareto front and has following variable linkage function:

$$x^8 \leftarrow 1 + \frac{i}{|x^8|} \times (x^8 - l_i) - x^f \times (u_i - l_i) \quad \forall i = 1, \ldots, |x^8| \tag{1.31}$$

This function has the following objective functions:

$$\begin{aligned} f_1(x) &= x^f \ldots \left(1 + \sum_{j=1}^{M-1} c_{1,j} \times g(x^8)\right) \\ f_2(x) &= x^f \ldots (1 - x^f_{M-1}) \left(1 + \sum_{j=1}^{M} c_{2,j} \times g(x^8)\right) \\ &\vdots \\ f_{M-1}(x) &= x^f (1 - x^f) \left(1 + \sum_{j=1}^{M} c_{M-1,j} \times g(x^8)\right) \\ f_M(x) &= (1 - x^f) \left(1 + \sum_{j=1}^{M} c_{M,j} \times g(x^8)\right) \end{aligned} \tag{1.32}$$

with

$$c_{i,j} = \begin{cases} 1, \text{ if } i = j \\ 0, \text{ otherwise} \end{cases} \tag{1.33}$$

and

$$\begin{aligned} g(x^8) &= \frac{1}{n_k} \sum \frac{\eta_6(x^8)}{|x^8|} \\ g(x^8) &= \frac{1}{n_k} \sum_{i,j}^{n_k} \frac{\eta_5(x_{i,j})}{|x^8|} \\ &\forall k = 1, \ldots, \frac{M}{2} \end{aligned} \tag{1.34}$$

i, j |

1.6.5 LSMOP5

This is an unimodal problem with a fully separable fitness landscape. It generates a nonlinear Pareto front and has following nonvariable linkage function:

$$\underline{i}\ x^8 \leftarrow 1 + \cos\ 0.5\pi$$

$$\times (x^8 - l_i) - x^f \times (u_i - l_i)$$

$$|x^8| \quad i \quad 1$$

(1.35)

This function has the following objective functions:

$$\begin{aligned}
f_1(x) &= \cos\left(\frac{\pi}{2}x^f\right)\cdots\cos\left(\frac{\pi}{2}x^f\right)\cos\left(\frac{\pi}{2}x^f\right)\left(1+\sum_{j=1}^{M}c_{1,j}\times g_j(x^8)\right)\\
f_2(x) &= \cos\left(\frac{\pi}{2}x^f\right)\cdots\cos\left(\frac{\pi}{2}x^f\right)\sin\left(\frac{\pi}{2}x^f\right)\left(1+\sum_{j=1}^{M}c_{2,j}\times g_j(x^8)\right)\\
&\cdots\\
f_{M-1}(x) &= \cos\left(\frac{\pi}{2}x^f\right)\sin\left(\frac{\pi}{2}x^f\right)\left(1+\sum_{j=1}^{M}c_{M-1,j}\times g_j(x^8)\right)\\
f_M(x) &= \sin\left(\frac{\pi}{2}x^f\right)\times\left(1+\sum_{j=1}^{M}c_{M,j}\times g_j(x^8)\right)
\end{aligned} \tag{1.36}$$

$$c_{i,j} = \begin{cases} 1, \text{ if } i = j \text{ or } j = i+1\\ 0, \text{ otherwise}\end{cases} \tag{1.37}$$

with

and

(1.38)

$$\eta_{i,j} = \left| \frac{i,j}{2^k} \right|$$

$$2k\ i$$

$$\square k = 1, \ldots, \quad M$$

$$_2 \quad 1 \qquad\qquad _2 \quad M-2 \qquad\qquad _2$$

$$\ldots$$

$$\overline{_2} \quad 1 \qquad\qquad j=1$$

1.6.6 LSMOP6

$$\bar{\quad}$$

$$\bar{\quad}$$

$$M-1 \qquad j=1 \qquad {}_2 \quad j \tag{1.40}$$

$$\bar{\quad}$$

$$\bar{}_M \quad j$$

$$x^8_i \leftarrow 1 + \cos 0.5\pi$$

$$x^8_i = 1, \ldots, \quad |x^8| \times (x^8_i - l_i) - x^f_1 \times (u_i - l_i)$$

$$f_1(x) = \cos\left(\frac{\pi}{x^f}\right)\cdots\cos\left(\frac{\pi}{x^f}\right)\cos\left(\frac{\pi}{x^f}\right)$$

$$1 + \Sigma_M$$

$$\square \qquad \mathbf{2} \quad 1 \qquad c_{1,j} \times g \qquad (x^8)$$

$$\square f_2(x) = \cos\ \frac{\pi}{} x_{\frac{\pi}{}} x^{f_{\cdot}} \cdot \cos\ \frac{\pi}{} x^{f} \quad \sin \qquad \sum_{1}^{M}$$

$$c_{2,j} \times \quad g$$

(x^8)

$$\square f_{M-1}(x) = \cos\left(\frac{\pi}{\sum_M} x^f\right) \sin\left(\frac{\pi}{+} x^f\right) \quad 1$$

$c_{M-1,j} \times g$

(x^8)

□ 2 1 2 2

$$f_M(x) = \sin\left(\pi x^f\right) \times \left(1 + \sum_{j=1}^{M} c_{M,j} \times g\left(x^8\right)\right)$$

with

$$c_{i,j} = \begin{cases} 1, & \text{if } i = j \text{ or } j = i + 1 \\ 0, & \text{otherwise} \end{cases} \tag{1.41}$$

and

$$g(x^8) = \sum_{k=1}^{M} \eta_3(x^8)$$

$$g(x^8) = \frac{1}{n_k} \sum_{i,j} \frac{\eta_2(x_{i,j}, 8)}{|x^8|} \tag{1.42}$$

$$2k-1 \quad \sum_{j=1}^{n_k} \frac{\eta_2(x_{i,j}, 8)}{|x^8|}$$

$$i, j = i, j \mid$$

$$\eta_{1,k}, \frac{1}{2}$$

$$2k\ i$$

$$k = 1, \ldots, M$$

1.6.7 LSMOP7

This is a multi-modal problem with a mixed separable fitness landscape. It generates

$$2^1 \quad 2^{M-2} \quad 2^{M-1} \quad j=1 \quad \frac{}{2}\ j$$

...

a nonlinear Pareto front and has following nonvariable linkage function:

$$x_i^8 \leftarrow 1 + \cos(0.5\pi$$

$$\times (x^8 - l_i) - x^f \times (u_i - l_i)$$

$$|x^8| \quad i \quad 1$$

(1.44)

$$1 + \Sigma_M$$

$$f_1(x) = cos \ \frac{\pi}{} x^f \cdot\cdot cos \ \frac{\pi}{} \qquad x^f \qquad cos \ \frac{\pi}{} \qquad x^f$$

$c_{1,j} \times g$ (x^8)

□ **2** 1

$$f_2(x) = \cos\left(\frac{\pi}{2} x_1\right) \ldots \cos\left(\frac{\pi}{2} x_{M-1}\right) \sin\left(\frac{\pi}{2} x_M\right)$$

$$c_{2,j} \times g$$

$$f_{M-1}(x) = \cos\left(\frac{\pi}{2} x^f\right) \sin\left(\frac{\pi}{2} x^f\right) + \sum_{M} 1 \quad (x^8)$$

□ 2 1 2

$j=1$

$M-1j$

$$f_M(x) = \sin\left(\pi x^f\right) \times \left(1 + \sum_{j=1}^{M} c_{M,j} \times g\right) \quad (x^8)$$

$$M_{i,j} = \begin{cases} 1, & \text{if } i = j \text{ or } j = i + 1 \\ 0, & \text{otherwise} \end{cases} \tag{1.45}$$

and

$$\eta_g(x^8) = \sum_{j=1}^{n_k} \eta_6(x^{8\,j})$$

$2k-1$ 8 $,j$

i

8

(1.46)

$$\sum_{j=1}^{n} ,$$

i, j |

1.6.8 LSMOP8

This is a problem with mixed modality and a mixed separable fitness landscape. It generates a nonlinear Pareto front and has following nonvariable linkage function:

$$x^8 \leftarrow 1 + \cos(0.5\pi \times (x^8 - l_i) - x^f \times (u_i - l_i)$$

$$i = 1, \ldots, |x^8| \qquad i \qquad |x^8| \qquad 1 \tag{1.47}$$

This function has the following objective functions:

$$f_1(x) = \cos\left(\frac{\pi}{2} x^f \cdots \cos\left(\frac{\pi}{2} x^f\right) \cos\left(\frac{\pi}{2} x^f\right)\right) (x^8) + \left(1 + \sum_{j=1}^{M} c_{1,j} \times g_j\right)$$

$$f_2(x) = \cos\left(\frac{\pi}{2} x^f\right) \cdots \cos\left(\frac{\pi}{2} x^f\right) \sin\left(\frac{\pi}{2} x^f\right) (x^8) + \left(1 + \sum_{j=1}^{M} c_{2,j} \times g_j\right) \tag{1.48}$$

$$\ldots$$

$$f_{M-1}(x) = \cos\left(\frac{\pi}{2} x^f\right) \sin\left(\frac{\pi}{2} x^f\right) (x^8) + \left(1 + \sum_{j=1}^{M} c_{M-1,j} \times g_j\right)$$

$$f_M(x) = \sin\left(\frac{\pi}{2} x^f\right) \times \left(1 + \sum_{j=1}^{M-1} c_{M,j} \times g_j\right) (x^8)$$

with

$$c_{i,j} = \begin{cases} 1, \text{ if } i = j \text{ or } j = i + 1 \\ 0, \text{ otherwise} \end{cases} \tag{1.49}$$

and

$$g_j(x^8) = \frac{1}{n_k} \sum_{i=1}^{n_k} \eta_5(x^8_{i,j})$$

$$g_j(x^8) = \frac{1}{n_k} \sum_{i=1}^{n_k} \left(1 - \cos\left(\frac{\eta_1(x^8_{i,j})}{|x^8|} \times 2k - 1\right)\right) \tag{1.50}$$

$$\eta_{i,j} = \left| i, j \right|$$

$$2k\, i$$

$$k = 1, \ldots, M$$

1.6.9 LSMOP9

This is a problem with mixed modality and a fully separable fitness landscape. It generates a disconnected Pareto front and has following nonvariable linkage function:

$$x_i^8 \leftarrow 1 + \cos 0.5\pi$$

$\times (x^8 - l_i) - x^f \times (u_i - l_i)$

$|x^8| \quad i \quad 1$

(1.51)

This function has the following objective functions:

$$\begin{aligned} f_1(x) &= x_1 \\ f_2(x) &= x_2 \\ &\dots \\ f_{M-1}(x) &= x_{M-1} \\ f_M(x) &= (M - \Sigma_{M-1}) \times (2 + \Sigma_M) \times g(x_M) \end{aligned} \tag{1.52}$$

with

$$g(x_M) = 1 + \frac{9}{|x_M|} \sum_{x_i \in x_M} x_i$$

M,j $M\ j$

$$\frac{}{} \quad \frac{i,j}{|x\ |}$$
$$n_k \quad \frac{\eta_6(x_{i,j})}{|x^8}$$
$$\frac{k}{2}$$

$$c_{i,j} = 1 \tag{1.53}$$

$$g \quad (x^8) = \ ^1$$

and

$$\square g$$

$$\sum_{nk}$$

$$\sum_{j=1}^{i} n_k$$

$$n, \quad j=1$$

$$2k \; i$$

***i*, *j* |**

$\square k = 1, \ldots, M$

In the poultry industry, the challenges are considerably greater. Since it is a livestock type of material, the supply chain must be at the same time very large and very tight, as seen in Flanders and Gillespie (2015). Large since the livestock production involves genetics, feeding, breeding and growth control up to the chicken grandparents and tight since during the whole process there are very strict sanitary controls with the ration, water, diseases, effluents, vaccination and temperature, for example. Furthermore, due to the demand from various markets and to minimize the waste, almost all the parts of the chicken must be industrialized.

Considering these challenges, the production planners need to create production plans for a given product line and distribute it accordingly along with its plants. Since the variables are prone to changes over the time (depending, for example, on the market seasonal demands and the plant availability), the production plans are usually changed periodically.

In order to automate the creation of such plans, many industries use software algorithms where proposed solutions are generated based on these ever-changing variables. In the poultry industry, these algorithms are exposed to a larger degree of complexity since the number of materials to be produced are higher than the number of materials in the other industries summed with the higher volatility of the market.

However, many algorithms used by these industries will attempt to optimize a single objective - the profits. Of course this poses as an additional problem because, as exposed earlier, the estimated profits are not true since they are *theoretical* profits - the real world situations would then attenuate these profits. Since this consideration is not taken in account by simple, single-objective algorithms, it is likely that there are better alternatives with slightly lower theoretical profits but, at the same time, with higher reliability. From these alternatives the solution designer should then be able to do the choices accordingly.

The proposed MOP should have two objectives - reliability and profitability. Therefore, it is expected to have a Pareto front generated with the best solutions found considering the tradeoffs between both objectives and shown graphically for comparison purposes. Although it seemingly is not complex since it only have two objectives it does have on the other hand a high amount of constraints and variables which makes the work difficult for a given multi-objective algorithm. In this MOP, the reliability is a percentage of how much a given plant historically covers the production plans and the weighted profitability Wr is determined as the equation (1.55), where is determined by the sum of the products of the reliability R of the plant p by the profitability P found for the same plant p for all the four weeks and all the plants. This weighted profitability should show that it is less than the value returned for the profitability by any solution. Therefore, proposed solutions with higher reliability rates are expected to have less differences between the theoretical

and the weighted profitabilities even though both of them should be lesser than the ones found by the solutions geared towards the profitability in mind.

$$Wr = \sum_{i=1}^{4} R_ip * P_ip \tag{1.55}$$

The problem shown in Monteiro and Reynoso-Meza (2017) was built around the following rules:

1. There are nine plants;
2. All of them must be used. Also, all of the plants have a *reliability* rate assigned to them according to historical data. 6 plants were rated as 55.6% while the remaining 3 were rated as 77.8%;
3. A minimum quantity must be assigned for production for 124 materials according to the market demand. This quantity should be distributed along four weeks;
4. Each material belongs to one of the 11 available material groups and each material has a different value for sale;
5. There are varying production availabilities depending on the material group, plant and week. In other words, a given plant may offer varying maximum production capabilities for a given material group depending on the week. Also, one plant can hold the production of more than one material group which, in turn, may distribute the production along one or more materials within a given material group;
6. If a plant is able to produce stock from a material group it does not necessarily mean that all of the materials under that group are allowed for that plant. Therefore, alist of allowed materials per plant was provided as well.

Since in the provided data the sum of the monthly supply and demand were equal, naturally there is less freedom to explore the possible solutions - i.e. if the algorithm cannot take advantage of the most profitable materials in the expense of the lesser ones and manufacture more of them, the maximum it can do is to try to push the productionof the most profitable materials to the most reliable plants.

Based on all the aforementioned information, the MOP has the following characteristics:

1. 2 objectives: maximize the profitability and maximize the reliability;
2. 2032 variables - 508 per week. The variables are the production assigned for each material, for each plant. Most of the variables are integers with the remainder being floating point values.

3. 488 constraints - 124 of them are the market demands and all the others are theplant capacities for each week.

These values were retrieved out of the ERP (Enterprise Resource Planning) system responsible of collecting and managing this information. The ERP follows strict Audit and Compliance rules and, therefore, the information is as close as possible to the reality. The data is also managed by the Planning team inside the company, a specialized teamcomposed of 10+ analysts.

2 PRELIMINARY CONTRIBUTIONS

During the research period, other contributions were created as part of the direct path towards the algorithm proposed by this document. Although they were not created independently from the main research, it is important to highlight they were created as side objectives and results of it. The poultry dataset mentioned in the Section 1.6.10, for example, gave the grounds to test whether it was feasible to run a multi-objective optimization algorithm for this and other similar problems which, in turn, resulted in the COBEM 2017 paper.

2.1 Poultry dataset

The poultry dataset, as shown in the Section 1.6.10, was a dataset originated out of a company real-world situation. The complexity of this problem was important to test a multi-objective optimization implementation for it - specially considering the company, at that given moment, did not had implemented any multi-objective optimization scenarios. This dataset had the following characteristics:

1. 2 objectives: maximize the profitability and maximize the reliability of the production plan;

2. 2032 variables - 508 per week. The variables are the production assigned for eachmaterial, for each plant.

3. 488 constraints - 124 of them are the market demands and all the others are theplant capacities for each week.

While the dataset creation proved itself successful as in enabling the evaluation of a multi-objective optimization algorithm for it, after the results shown in the Section 2.2 the company itself found additional opportunities for improvement from a data quality standpoint. For instance, the reliability grades retrieved from the plants were very similar one from the other, which could result in suboptimal business decisions. Therefore, while this dataset could be used in one of the preliminary contributions, it could not be adopted in the main research afterwards since it was under business review for a newer, improved version.

2.2 Results published in COBEM 2017

The multi-objective optimization algorithm tested in a real-world scenario using the poultry dataset mentioned in the Sections 1.6.10 and 2.1 was presented in the COBEM 2017 congress (24th ABCM International Congress of Mechanical Engineering), held in Curitiba, Paraná, Brazil and available in Monteiro and Reynoso-Meza (2017).

Considering the quantity of objectives and solutions generated, instead of mathematical tools or other visual strategies such as radar plots or parallel coordinates, simple two-dimensional scatter plots were chosen instead. These plots uses the median reliability rates and the sum of the profitability for the suggested production distribution in its axes. The points in the plot shows the position of the solutions relative to both axes. If the number of proposed solutions were bigger to a point that the decision-making was difficult to the problem designer other strategies could be used instead.

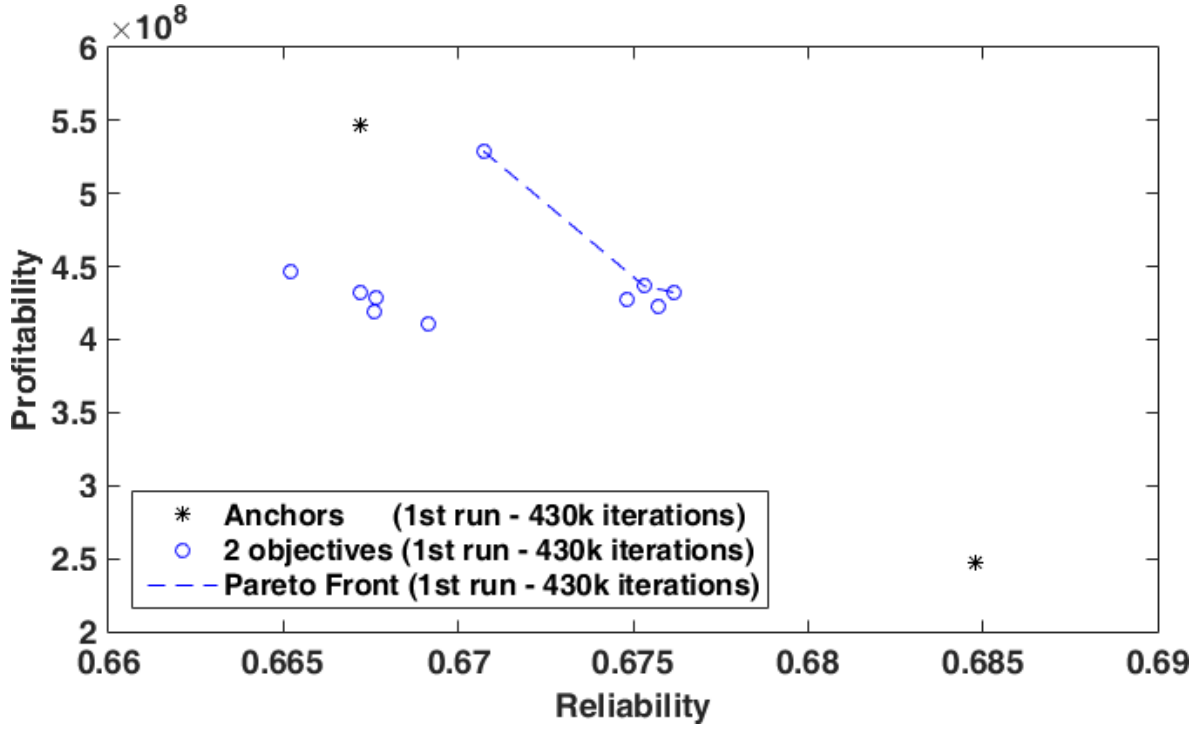

Figure 12 – Results of the first run done with 350000 iterations. The first plot shows the tradeoffs between both objectives while the second plot shows the comparison between the theoretical maximum profits and the more realistic profits takingin account the reliability rates found for each solution.

As seen in the first plot of the Fig. 12, below, running with 430 thousand iterations proved that the available window to optimize the reliability was very small, with less than 2% of difference between both anchor points. However, by maximizing the reliability the profitability would be too affected, with a financial difference of approximately $3 * 10^8$, or more than 50%, from the anchor point belonging to the profitability maximization. Even

after considering the difference caused by the changes in the reliability the variance in the weighted profitability would still be too high, as seen in the bottom plot of the Fig. 12.

By choosing the best solution found (i.e. the only solution with more than $5 * 10^8$ in profitability found) it was possible to determine that, although there was in fact an inverse relationship between profitability and reliability, the differences were negligible. By choosing this solution instead of the solution found in a single-objective approach - i.e the profitability anchor point - there was an increase of approximately 0.3% in the reliability with a decrease in the profits of around 3.2% or, when the own reliability is taken in account in the determination of the weighted profitability, the decrease is of approximately 2.7%. Also, since all but one solution had expected profits closer to the maximum profits and the difference in the reliability is small in percentage, it is possible to verify that more accurate solutions could be generated if more iterations were provided.

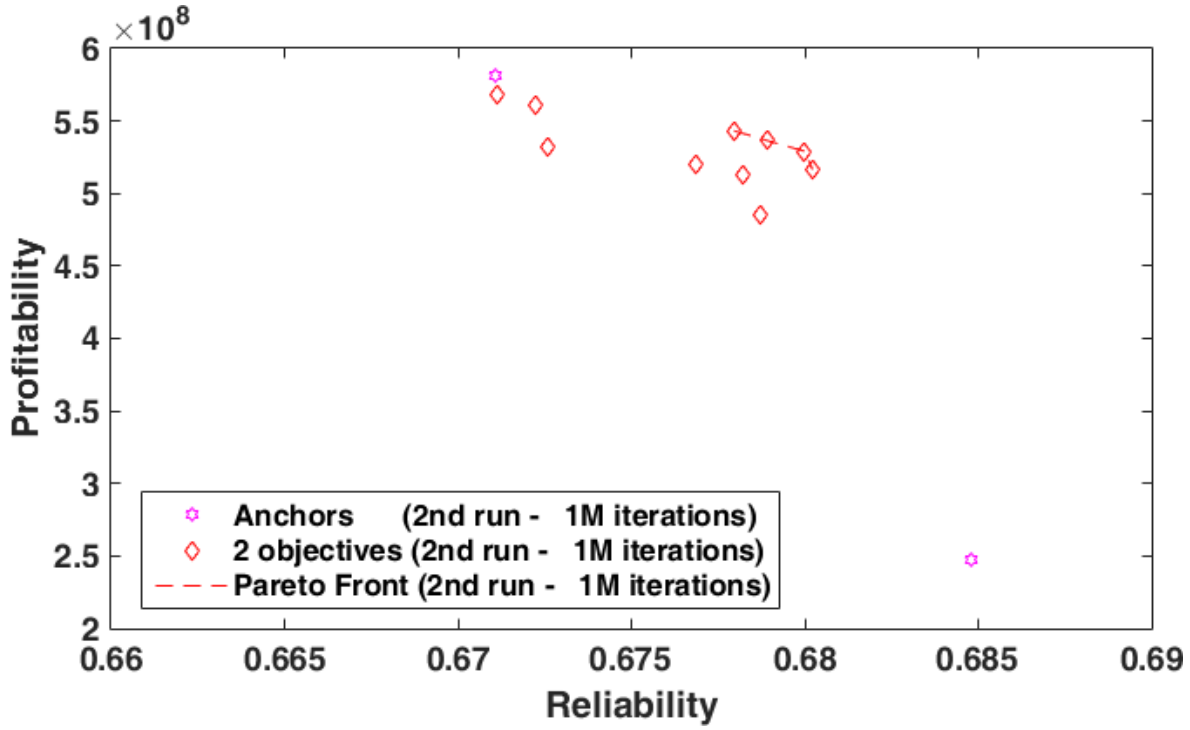

Figure 13 – Results of the second run done with 1 million iterations. The first plot shows the tradeoffs between both objectives while the second plot shows the comparison between the theoretical maximum profits and the more realistic profits takingin account the reliability rates found for each solution.

In order to test this hypothesis, the second run, with 1 million iterations took place. As seen in first plot of the Fig. 13, the results were more evenly spread as expected. The profitability anchor point had higher values for both objectives, with all the solutions following the same pattern. However, the reliability anchor could not be improved since it had already reached its best value. Also, as shown in the second plot, the same situation of the first run repeated itself - by choosing almost any solution the tradeoffs would be

negligible - although the reliability improvement could be a higher, but still under 1%, the reduction in the weighted profitability would stay between 2% and 3% for the best solutions within the Pareto front.

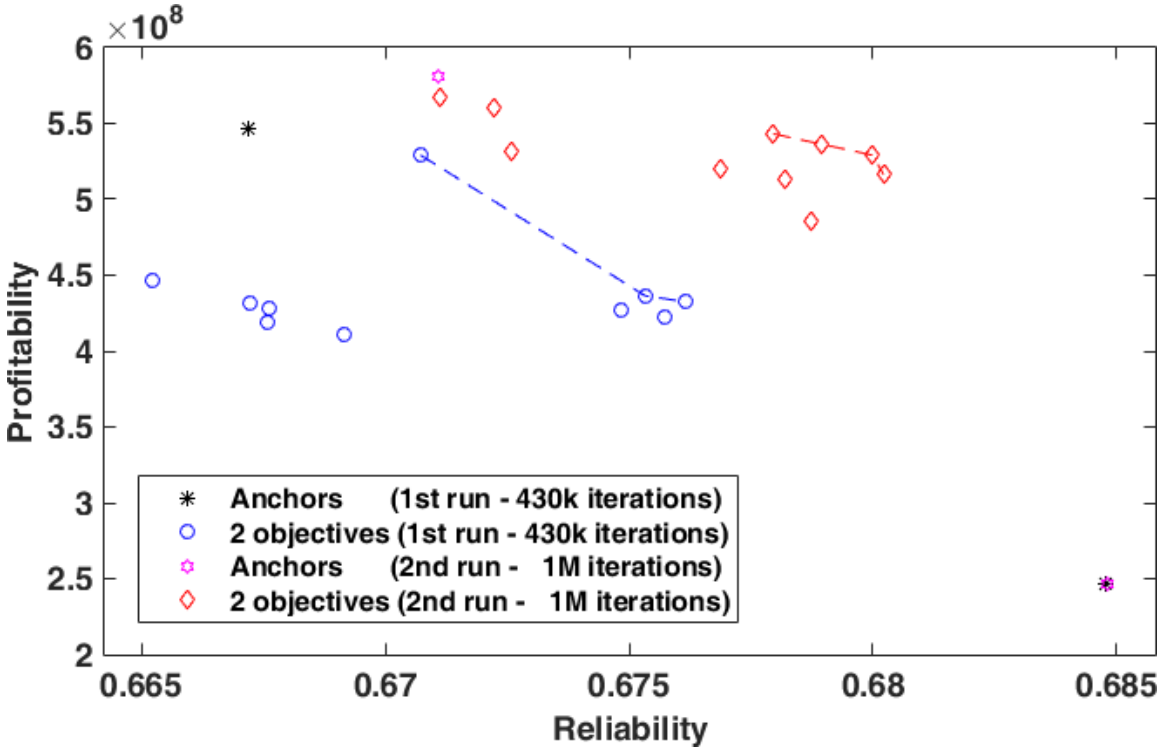

Figure 14 – Comparison between both runs. The second run achieved noticeably betterresults.

In this MOP, it is also important to note a pattern associated with the number of iterations used. As seen in the Fig. 14, the results had better solutions with their general positions located both to better reliability and profitability, mainly due to the increased number of iterations. As seen in the second plot of the Fig. 14, for example, the anchor point for the profitability in the first run is worse than at least three solutions found forthe second run from a profitability standpoint.

On the other hand, the maximum reliability possible could not be improved due to the limits placed by the problem. Since the losses in the profitability would be too high for small improvements in the reliability, for both runs the solutions would be more oriented towards the profitability anchor.

This problem, using an i7-3770K desktop, took around 30 hours to properly execute in a single run in the simplest scenario. Under business reviews, while the company deemed the results as acceptable, the time taken was too high to be feasible in daily business situations - specially considering the algorithm would likely take more time to execute using off-the-shelf laptops with less powerful specifications. This situation was one of the main motivators behind the main research.

3 PROPOSAL

Considering the differences between single and multi-objective optimization as well as the background of the current research in the form of preliminary contributions, the effective challenge is given in the form of a multi-objective optimization algorithm capable of solving large-scale problems with low computer resource usage. As such, it is relevant to highlight the importance of the preliminary contributions in providing the grounds for the current proposal, even though the original dataset (poultry) could not be used due to company internal data reviews. However, in order to validate the proposal, other mathematical problems with similar features from a complexity standpoints are adopted instead. The algorithm proposed in this document was also presented in the HPOI 2018 conference (International Conference on High-Performance Optimization in Industry), part of the 21st International Multiconference of the Information Society held in Ljubljana, Slovenia and available in Monteiro and Reynoso-Meza (2018). As a direct result from the COBEM 2017 presentation and proposal, a hybrid multi-objective optimization evolutionary algorithm built towards large-scale problems and with low computer resource usage was created and evaluated in the scenarios shown in the Section 4.

3.1 Context

The objective of this research, considering all the topics presented in the previous section, is to propose a new optimization algorithm that meets the following criteria:

- The algorithm must solve multi-objective problems;
- The algorithm must be targeted towards large-scale, multi-variable problems;
- The algorithm must be able to attempt to quickly reach the real Pareto front;
- The algorithm must not be resource-intensive;
- It is easy to evaluate, adapt and port.

Considering the big picture, there are multiple multi-objective algorithms already available, specially in the EMO field Coello et al. (2007) However, they are mostly tested against 2-3 objective problems instead of 5+ or 10+ objectives, for example. Furthermore, the number of variables presented in some datasets is small (≤ 100). It is acknowledged, though, that this is a well-explored field with many variants, each with their trade-offs. For this reason, the framework shown in the Section 1.6 which in turn allows a range of known algorithms will be used in this research.

In that sense, as shown earlier there are also some techniques targeted to large-scale optimization. The increased number of variables and objectives present itself as a harder challenge due to the computational power required to attempt to draw the feasible space and, therefore, the feasible solutions based on the allowed variable values are considerably harder to identify - by extension, to improve them until they reach the Pareto front. There are cooperative co-evolutionary strategies looking forward to tackle down this issue. Since they use a EMO algorithm behind the scenes, it was one of the alternatives considered to be the basis of the strategy presented by this document.

Another strategy considered was the implementation of a hybrid algorithm. From the start it was considered as a viable option to be taken considering it is easiness to test, adapt during the development phase and possibly port to other languages - specially considering one of its main intended use is to be used in corporate environments. Therefore, it is not possible to guarantee said environments will always have MATLAB® available. Furthermore, the proposed hybrid algorithm offered good results in the tests.

3.2 Overview

The algorithm hereby proposed is a hybrid multi-objective optimization strategy intended for large scale problems focused on performance in corporate environments. It is important to mention this research heavily focused on the multi-objective optimization (MOO) algorithm (explained in the Section 1.2.2) instead of the multi-criteria decision making (MCDM) since it is understood that the output of the MOO phase can be used with some MCDM strategies later on in the decision-making process. Furthermore, the intention of the current research is more focused in *generating* solutions instead of ensuring any of them is chosen in the end of the process - mostly because the decision-making process has larger human involvement.

The high-level structure of the algorithm is shown in Algorithm 6 and will be explained in more detail in the subsequent sections. The lines 1 and 2 are in charge of generating a n set of random solutions as well as evaluating themselves based on their Pareto front values. Then, for each generation it will store the previous generated solutions (lines 3 and 4). Based on these solutions it will generate offspring and join them with all the other solutions (lines 5 and 6). Then, from all of these solutions the algorithm will only select the best ones and replace them with their locally improved counterparts (lines 7 to 9). Finally, it will rank all of these solutions again (line 10). Finally, some of the solutions will be pruned before moving to the next generation (line 11).

In the aforementioned algorithm all of the underlined key components are shown in more detail below.

Algorithm 6: New algorithm

Data: design space, objective vector, parameters (Section 3.2.1)
Result: the Pareto front approximation
1 generate *n* random solutions;
2 evaluate the solutions;
3 rank the solutions through a dominance filter (Algorithm 8);
4 for *each generation until it reaches the stopping criteria* **do**
5 store the previous generated solutions as well as their front values;
6 generate and evaluate offspring through tournament, recombination and mutation (Algorithm 9);
7 join the offspring to the other solutions;
8 partially rank the joined solutions through a dominance filter (Algorithm 8);
9 locally improve the best solutions (Algorithm 13);
10 replace the best solutions with the locally improved solutions;
11 rank all the joined solutions through a dominance filter (Algorithm 8);
12 prune the number of solutions generated (Algorithm 15).
13 end

3.2.1 Parameters

In order to best suit the different algorithm needs a set of parameters is used. Considering the current implementation this set is loaded before the execution of the algorithm itself. The parameters used are as follows:

1. Problem bounds (design space);
2. Maximum number of generations allowed;
3. Mutation α probability (from 0.0 to 1.0) (Algorithm 11);
4. Mutation x Recombination probability (from 0.0 to 1.0) (Algorithm 9);
5. Number of objectives (of the problem);
6. Number of variables (of the solutions);
7. Initial population size;
8. Maximum front size per front level;
9. Maximum number of generations in sequence without improvements before stopping the algorithm (Algorithm 7);
10. Percentage used to determine if a generation was improved in relation to the previousone (from 0.0 to 1.0) (Algorithm 7);
11. Maximum non-domination level to be considered between generations (Algorithm 15);

12. Maximum non-domination level to be considered for the local search (Algorithm 15);
13. Number of random neighbors for the local search *(optional)* (Algorithm 13);
14. Flag responsible to whether the algorithm should forcefully prune each generation (Algorithm 15).

This set of parameters is used considering the requirement of improving the algorithm performance depending on the problem used as well as providing means of tuning the algorithm itself in order to ensure it will get to the true Pareto front.

The *problem bounds* as well as the *number of variables*, although quite redundant (from the characteristics of the bounds it is possible to know the number of variables), are identified as separate parameters for the sake of algorithm speed improvement since the algorithm would only need to access one integer instead of deriving that information out of the bounds vector every time it needs to use that information.

The *number of objectives* and the *initial population size* are self-explanatory - they keep track of how many objectives the problem being tested has as well as its initial, random population size. A population too high will lead to a large number of offspring, leading in turn to poorer performance. On the other hand, an initial population too small will likely lead to a higher amount of generations required in order to arrive to the true Pareto front.

The *maximum number of generations*, the *maximum number of generations in sequence without improvements before stopping the algorithm* and the *percentage used to determine if a generation was improved in relation to the previous one* compose the algorithm stopping criteria. At least one of these criteria must be met before stopping the algorithm altogether. The first one limits how many generations the algorithm will support for a given problem. The second and the third criteria (from now on named as *MaxGenWithoutImprovement* and *PercentageImprovement* respectively) are used together:

$$ObjectivesImproved = \sum \frac{1 - \frac{\min(currentGenFront)}{\min(previousGenFront)}}{} \geq PercentageImprovement \quad (3.1)$$

Equation (3.1) (adapted for minimization problems only, but able to be modified to also suit maximization objectives) determines how many objectives had been improved. The percentage of improvement from the current to the previous generation is determined considering the minimum value found for both generations from each one of the objectives. In the end, *ObjectivesImproved* would store how many objectives would be indeed

improved. After finishing a given generation, the Algorithm 7 tests if the current generation had actually improved itself over the previous one, as shown here:

Algorithm 7: Check for improvements

Data: solutions from the current generation and from the past generation
Result: true if the current generation improved over the past one and false if not

```
1 if any objective was improved equation (3.1) then
2 |   LocallyImprove = true;
3 |   CountGenW ithoutImprovement = CountGenW ithoutImprovement + 1 ;
4 else
5 |   LocallyImprove = false;
6 |   CountGenW ithoutImprovement = 0.
7 end
```

Two variables are important in the global algorithm - *LocallyImprove* is a flag in charge of enabling the local search for the next generation. As explained in more detail in the Section 3.2.4, the implementation of local search shown important improvements in the preliminary tests over not using it at all. On the other hand, it heavily made use of computational resources to the point of making its use in all of the generations counterproductive. For this reason, instead of using a probability-driven parameter to locally improve the offspring a given generation, the Algorithm 7 is the one in charge of triggering it - if it detected the current generation improved over the previous one as is, then the chances it is in the right path are higher. However, if it detected no improvements in the current generation happened, the local search might be required to get out from a theoretical plateau, if it exists. The Algorithm 7 should only use as input the solutions part of the first γ non-dominating levels, where γ is the value set by the *maximum non- domination level to be considered for local search* parameter. Each of these solutions will generate a number of η random neighbors - if the *number of random neighbors for the local search* parameter had been set, η will be equal to that parameter. Otherwise, η will be equal to the number of variables in the problem.

CountGenWithoutImprovement is the other variable. In the start of the global algorithm it would be initialized as zero and it is a counter used to keep track of how many subsequent generations did not improve. The global algorithm will then stop as soon as its value equals *MaxGenWithoutImprovement*.

The *mutation x recombination probability* is the ratio (between 0.0 and 1.0) on which a given set of solutions will be used for mutation with the remainder used for recombination. On an example, if this ratio is set to 0.6, 60% of the solutions will be used for mutation while the other 40% will be used for recombination. Inside the mutation algorithm the *mutation α probability* is used to pick the variables to be mutated. More onit on the Section 3.2.3.

The *maximum non-domination level to be considered between generations*, *maximum front size per front level* and *flag responsible to whether the algorithm should forcefully prune each generation* determines the depth and width of a given generation to be transported from one generation to its successor. Let us consider that a given generation ended with 400 solutions, distributed in 7 different non-domination levels. If *maximum non-domination level to be considered between generations* is set to 4, only the solutions that are members of the first four non-domination levels will be transported to the next generation, with the remainder being removed. On the other hand, if this parameter is set to 10, all of the solutions will be transported to the next generation since it would support up to 10 non-domination levels and only seven were available at that time. Moreover, if the *flag responsible to whether the algorithm should forcefully prune each generation* is set to *true*, all of the solutions part of the non-domination levels that will be transported to the next generation will be scanned. If any of those non-domination levels have more solutions than the maximum amount set in *maximum front size per front level* the excesses will be pruned out according to the algorithm shown in the Section 3.2.5.

3.2.2 Dominance Filter

The dominance filter is one of the key elements of the algorithm since it is in charge of determining the non-dominating levels of a given set of solutions. Its logic is described as shown in the Algorithm 8.

This algorithm basically compares all the unassigned solutions amongst themselves. From these solutions, the non-dominated ones are assigned with the current non-domination level, starting with 1 and increasing afterwards. Then, another comparison starts, again considering only the non-dominated solutions iteratively until all the solutions have been assigned. Understandably, one can conclude it results in smaller solution pools as the algorithm passes through the non-domination levels.

Nevertheless, depending on the scenario it is not required to know all the non-domination levels - for example, looking through the Algorithm 6 it is possible to notice that one of the steps mentioned *partially ranks* the solutions. In other words, it is only needed to know the solutions in the first γ non-domination levels, as set by the *maximum non-domination level to be considered for local search* parameter (section 3.2.1) since they will be used in the local search. γ is assigned to the *MaximumAllowedLevel* variable in this case, causing an early finish (and, therefore, faster performance when it is needed) of the new algorithm as soon as it reaches that value. Also, it is important to highlight the for-loop is parallelized in this implementation leading to further improvements as far as memory consumption and CPU (Central Processing Unit) usage are concerned.

On the other hand, in the other scenarios mentioned in the Algorithm 6 when this same dominance filter is called it is also possible to assign to

MaximumAllowedLevel

Algorithm 8: Dominance Filter

```
Data: the Pareto front approximation, MaximumAllowedLevel (optional)
Result: the non-domination levels for all the Pareto front approximations
1  CurrentNonDominationLevel = 1;
2  if MaximumAllowedLevel is not informed then
3  |  MaximumAllowedLevel = ∞;
4  end
5  while there are solutions unassigned to any non-domination level do
6  |  get all the unassigned solutions;
7  |  for all the unassigned solutions in the Pareto front approximation do
8  |  |  if the current solution is non-dominated by all the unassigned solutions
   |  |  then
9  |  |  |  assign CurrentNonDominationLevel to the current solution;
10 |  |  end
11 |  end
12 |  if CurrentNonDominationLevel ≥ MaximumAllowedLevel then
13 |  |  assign CurrentNonDominationLevel + 1 to the remaining
   |  |  unassigned solutions;
14 |  end
15 |  CurrentNonDominationLevel = CurrentNonDominationLevel + 1.
16 end
```

the *maximum non-domination level to be considered between generations* parameter since this will be the threshold used by the algorithm when handling the solutions from one generation to its successor, meaning it is also feasible to experience improvements in memory and CPU usage by limiting the non-domination levels in these cases as well.

3.2.3 Tournament, mutation and recombination

The tournament, mutation and recombination are three steps done in sequence according to the following logic:

Algorithm 9: Tournament, mutation and recombination

Data: the Pareto front approximation

Result: mutated and recombined solutions

1. get the best solutions through tournament (Algorithm 10);
2. λ = *MutationProbability* $*$ *NumberOfBestSolutions*;
3. get the mutated solutions from the 1st to the λ-th best solutions (Algorithm 11));
4. get the recombined solutions from the (λ + 1)-th to the last best solutions (Algorithm 12).

According to this algorithm, the Pareto front (which includes all the non-domination levels) is used to generate offspring (i.e. the mutated and recombined solutions). The offspring is generated from the best solutions found from a tournament. These solutions -

which are in a random order - are separated into two groups with λ being the divider. λ is the product of the number of solutions in the output of the tournament and the *mutation x recombination probability* shown in the Section 3.2.1. λ therefore would be the index of the solution used as cutoff - if 100 solutions compose the output of the tournament and λ equals 60, then the first 60 solutions are used in the mutation while the last 40 solutions are used in the recombination. The tournament logic in itself is shown as follows:

Algorithm 10: Tournament

Data: a Pareto front approximation

Result: tournament winners

```
1  calculate the crowding distances for all the solutions;
2  create MaximumFrontSize * MaximumN onDominationLevelBetweenGenerations
   brackets;
3  foreach bracket do
4  |  randomly assign 2 Pareto front solutions for the bracket;
5  |  if both solutions have the same non-dominating level then
6  |  |  winner = solution with the largest crowding distance;
7  |  else
8  |  |  winner = solution with the smallest non-domination level;
9  |  end
10 end
```

The Algorithm 11 shows the mutation logic. There, *Random* is an array with a size equal to the number of variables with all of its values randomly generated between
0.0 and 1.0. Then, as shown in the equation (3.2), these numbers are compared against the α mutation probability (a parameter with a value between 0.0 and 1.0, as shown in the Section 3.2.1). All of the values below this threshold will have a logical *false* value (0) as a result while all of the values with the same value or above it will have the logical *true* value (1) assigned to it. Naturally, if a variable was not chosen to be mutated it will have the final value of zero while the variables chosen to be mutated will have a random value respecting both the upper and lower bounds.

$$Mutate = (Random < \alpha) * (LowerBounds + ((UpperBounds - LowerBounds) * Random)) \tag{3.2}$$

The Algorithm 12 shows the recombination logic. It will generate a number of children equal to half the size of the Pareto front approximation given to it. Each children is generated from two solutions of the Pareto front approximation according to the (3.3) equation. It is important to remember that the Pareto front approximation fed to this algorithm is based on some of the tournament winners and, as shown in the Algorithm 10,

Algorithm 11: Mutation

Data: a partial Pareto front approximation
Result: the mutated Pareto front approximation
1 foreach *solution of the Pareto front approximation* **do**
2 | *Random* = random array with values between 0.0 and 1.0;
3 | *Mutate* = equation (3.2);
4 | **foreach** *position in Mutate different from zero* **do**
5 | | change the value in the solution to the value in *Mutate* for the same index;
6 | **end**
7 end
8 evaluate the mutated solutions.

the order on which these winners are presented is random.

Algorithm 12: Recombination

Data: the Pareto front approximation
Result: recombined solutions
1 define as the *NumberOf Children* half the size of the Pareto front approximation;
2 foreach *integer i between 1 and NumberOf Children* **do**
3 | get the $(i * 2)$-th and $(i * 2) - 1$-th solutions;
4 | assign the *i*-th children according to the equation (3.3);
5 end
6 evaluate the recombined solutions.

$$Children = FirstSolution + (SecondSolution - FirstSolution) * Random \tag{3.3}$$

3.2.4 Local Search

The local search algorithm ensures the best solutions are forcefully improved before moving to the next generation. Therefore, the general idea is to minimize the number of generations by attempting to find better solutions beforehand through forced evolutions instead of just relying upon a standard genetic algorithm. As an expected result, there could be improvements in speed, convergence and efficiency in the general algorithm. The local search strategy employed in this proposal is shown in the Algorithm 13.

The *MaxNonDominationLevelForLocalSearch* parameter in the Algorithm 13 refers to the *maximum non-domination level to be considered for local search* parameter mentioned in the Section 3.2.1. This algorithm divides the solutions in two parts based on their non-domination levels. All of the solutions members of a non-domination level *up to* this threshold are added to the *improvedSolutions* matrix. Then, these solutions are processed through a Pareto improvement algorithm. These new, Pareto improved

solutions are also added to *improvedSolutions*.

Algorithm 13: Local Search

Data: the Pareto front approximation
Result: the improved Pareto front approximation
1 initialize *improvedSolutions*;
2 get all the solutions up to the *MaxN onDominationLevelF orLocalSearch*;
3 evaluate these solutions;
4 add all these solutions to *improvedSolutions*;
5 **foreach** *solution part of the non-domination levels up to MaxN onDominationLevelF orLocalSearch* **do**
6 | Pareto improve the current solution (Algorithm 14);
7 | add the Pareto improved solutions to *improvedSolutions*;
8 **end**
9 rank the best solutions in *improvedSolutions* through a dominance filter;
10 concatenate these solutions with the other solutions above the *MaxN onDominationLevelF orLocalSearch* threshold.

All of the solutions in *improvedSolutions* - i.e. the original solutions selected from *MaxN onDominationLevelF orLocalSearch* and their Pareto improved children - are processed through the dominance filter (Section 3.2.2). Only the best ones (the solutions in the first non-domination level) are picked and joined with all the other solutions that where not selected since their non-domination level was greater than *MaxN onDominationLevelF orLocalSearch*. This modified Pareto front approximation is then returned.

On the Pareto improvement, the Algorithm 14 shows how does it work. The *number of random neighbors for the local search*, an optional parameter shown in the Section 3.2.1 and represented by *NumberOf RandomN eighbors*, is used to improve the performance in scenarios where there are too many variables and this algorithm would take too long to process all of the variables.

For example - if a solution has 1000 variables and this parameter is not informed, this algorithm will create 1000 new solutions based on the original solution where the first solution assigned a new random value for the first variable while keeping all the other variables with the same value equation (3.4); the second solution assigned a new random value for the second variable while keeping all the other variables with the same value and so on. If 100 solutions are involved in the local search, at least 1e5 new solutions would be created as a result. On the other hand, if a solution had 15000 variables, considering the same 100 solutions as a result 1.5 million new solutions would be created. Considering the local search would happen more than once during the algorithm execution, this method - as seen in Liefooghe et al. (2012) and based on the

exhaustive neighborhood exploration,

would take a long time to run.

$$NewValue = (LowerBounds + ((UpperBounds - LowerBounds) * Random)) \quad (3.4)$$

When using the aforementioned parameter, on the other hand, it would greatly improve the speed and memory consumption. Bearing in mind the same case of 15000 variables but with the parameter *NumberOf RandomN eighbors* set to 1000, for each solution it would only pick 1000 out of 15000 variables randomly (instead of all the 15000). That way, the total amount of new solutions generated is dramatically decreased from 1.5 million (considering the same total of 100 solutions to be executed by the local search) to 1e5. This strategy is, according to Liefooghe et al. (2012), in practice a *partial neighborhood exploration* strategy.

Algorithm 14: Pareto improvement

```
   Data: a solution
   Result: Neighbors
 1 initialize the list of neighbors Neighbors;
 2 if NumberOf RandomN eighbors was informed then
 3 |   select NumberOf RandomN eighbors random variables;
 4 else
 5 |   select all the variables;
 6 end
 7 foreach one of the variables selected do
 8 |   copy the original solution;
 9 |   modify the variable value according to the equation (3.4);
10 |   evaluate the new solution;
11 |   add it to Neighbors;
12 end
13 replace Neighbors with only its anchors.
```

3.2.5 Pruning

At the end of each generation a situation might happen where the amount of solutions available per non-domination level is way too high for performance reasons. Or, in some cases where the solutions are spread almost linearly across the Cartesian plane, as the generation passes the amount of solutions in the first non-domination levels quickly increases causing performance impacts.

Seen in the 15, the *maximum non-domination level to be considered between generations* (*MaxNonDominationLevelBetweenGens*), *maximum front size per front level* (*MaxFrontSizePerFrontLevel*) and *flag responsible to whether the algorithm should forcefully prune each generation* (*ForcePrune*) parameters shown in the Section 3.2.1 are used to control the pruning.

Algorithm 15: Pruning

```
Data: the full Pareto front approximation
Result: the pruned Pareto front approximation
if ForcePrune is true then
    foreach non-domination level up to MaxNonDominationLevelBetweenGens do
        if NumSolutionsInCurrentLevel > MaxFrontSizePerFrontLevel then
            calculate the crowding distances (Algorithm 16);
            sort the solutions from the largest to the smallest distance;
            add the first NumberOfSolutionsInCurrentLevel solutions to the
              new Pareto front;
        else
            add all of the solutions in the current level to the new Pareto front
              approximation;
        end
    end
    overwrite the current, full Pareto front with the new front.
else
    remove all the solutions part of the non-domination levels above
      MaxNonDominationLevelBetweenGens.
end
```

If *ForcePrune* is not active (i.e. set to **false**), the algorithm will only remove from the Pareto front approximation all of the solutions that belong to non-domination levels above the threshold established by *MaxNonDominationLevelBetweenGens*. All the other solutions in the other non-domination levels will be kept in the front regardless of howmany solutions actually remain.

However, if *ForcePrune* is active (i.e. set to **true**), the algorithm will iterate through all of the non-domination levels up to *MaxNonDominationLevelBetweenGens* (thus not considering the solutions in the other levels in the same way as when *ForcePrune* is set to false). If the number of solutions in the current level (*NumSolutionsInCurrentLevel*) is greater than the value in *MaxFrontSizePerFrontLevel*, the algorithm will ensure the excesses will be pruned out by removing in that non-domination level the solutions with the smallest crowding distances through the 2-nearest neighbors approach, as shown in more detail in Chaudhuri and Dasgupta (2014).

The crowding distance determination logic, as described in the Algorithm 16, normalizes all the solutions for all the objectives and sorts them accordingly. For each objective it will assign ∞ to the solutions with the largest and smallest values. Then, all the other solutions will have the distance calculated by the equation (3.5). Since the solutions are sorted by the current objective, the neighbors immediately before and after it have the nearest values from its own. The difference between their values is then responsible for

helping in the determination of the crowding distance for the current solution.

$$Distance = Distance + (values(i + 1) - values(i - 1)) \quad (3.5)$$

Algorithm 16: Crowding Distance **Data:**
the Pareto front approximation **Result:**
the crowding distances

```
1  set the distances for all the solutions to zero;
2  foreach objective do
3  |  normalize the Pareto front approximation values for the current objective;
4  |  get the values of the current objective;
5  |  sort the values in ascending order;
6  |  for first and last sorted solutions do
7  |  |  distance = ∞;
8  |  end
9  |  i = 2;
10 |  while i < NumberOf Solutions do
11 |  |  set the distance to the solution in the i position according to the equation
   |  |   (3.5);
12 |  |  i = i + 1;
13 |  end
14 end
```

3.3 Strategies Previously Used

Considering the algorithm proposal, several modifications happened during the course of the algorithm implementation. The tournament, for example, originally employed the automatically selection of all the solutions from the first non-domination level plus a parameter p to determine the possibility to select the solutions in the following levels such as $p^{currentNonDominationLevel-1}$. However, as its results were not satisfactory, the implementation of the bracked system took place instead.

Also, the mutation logic solely included mixing the variables from two parents instead of using the logic shown in (3.2). However, it brought a slower pace on the improvements done through the generations.

Another strategy taken was the implementation of the Genetic Queued Pareto Local Search (GQPLS) strategy presented by Inja et al. (2014) for the local search due to its nature of preventing premature exclusion of dominated solutions. On the other hand, however, its implementation took too much time per call - therefore, it was dropped out from the final implementation.

It is relevant to highlight that in the overall implementation process several optimizations took place using the standard MATLAB® function calls and best programming

practices such as parallelization, matrix manipulation, code refactoring and parameter handling. As seen in the Section 4, these optimizations resulted in better resource usageas well as an acceptable performance.

4 RESULTS

The new, proposed algorithm as well as the other algorithms were tested in ani7-3770K desktop equipped with 16 GB RAM (Random Access Memory) with a dedicated
video card running Windows 10 Pro and MATLAB® R2015a. This desktop ran all the tests over the course of three weeks with one weekly system restart. In MATLAB® c , the start and end time of each algorithm were tracked (therefore storing the time taken for each execution) as well as the Pareto front approximation found for each execution. Outside MATLAB® , the Windows Performance Monitor (perfmon.exe, a known, built-in performance monitor tool in Windows) was used to track the performance and memory usage in MATLAB®. As such, it was able to properly track the resource usage by each algorithm.

4.1 Evaluation Methods

With the aforementioned tools the following characteristics were measured:

- The time taken for each execution (also named as *run*);
- The best Pareto front approximations (i.e. the first non-domination level) found for each algorithm;
- The hypervolume found for each algorithm considering the best Pareto front approximations found for them;
- The memory and processor usage for each algorithm.

Five different tests were executed:

- 1000 variables:

  9 problems (LSMOP1-LSMOP9 - more details in the Section 1.6);

  3 algorithms tested for each problem: gamultiobj; sp-MODE II; the new algorithm; 51 runs for each algorithm.

- 5000 variables:

  9 problems (LSMOP1-LSMOP9);

  3 algorithms tested for each problem: gamultiobj; sp-MODE II; the new algorithm; 51 runs for each algorithm.

- 15000 variables:

  9 problems (LSMOP1-LSMOP9);

  2 algorithms tested for each problem: gamultiobj; the new algorithm;51 runs for each algorithm.

- 30000 variables:

  9 problems (LSMOP1-LSMOP9);

  1 algorithms tested for each problem: the new algorithm;

  51 runs for each algorithm.

- 50000 variables:

  9 problems (LSMOP1-LSMOP9);

  1 algorithms tested for each problem: the new algorithm;

  51 runs for each algorithm.

From the problems with 15000 variables onwards not all of the three algorithms were tested since they started to trigger *out of memory* errors caused by their own memory allocation requirements. However, whenever applicable (i.e. some of the steps below were omitted should their algorithm was left unused due to the *out of memory* errors) the actions took place in the order shown below:

1. 51 runs for the new algorithm, LSMOP1;
2. 51 runs for gamultiobj, LSMOP1;
3. 51 runs for sp-MODE II, LSMOP1;
4. 51 runs for the new algorithm, LSMOP2;
5. 51 runs for gamultiobj, LSMOP2;
6. 51 runs for sp-MODE II, LSMOP2;
7. 51 runs for the new algorithm, LSMOP3;
8. 51 runs for gamultiobj, LSMOP3;
9. 51 runs for sp-MODE II, LSMOP3;
10. 51 runs for the new algorithm, LSMOP4;
11. 51 runs for gamultiobj, LSMOP4;
12. 51 runs for sp-MODE II, LSMOP4;

13. 51 runs for the new algorithm, LSMOP5;
14. 51 runs for gamultiobj, LSMOP5;
15. 51 runs for sp-MODE II, LSMOP5;
16. 51 runs for the new algorithm, LSMOP6;
17. 51 runs for gamultiobj, LSMOP6;
18. 51 runs for sp-MODE II, LSMOP6;
19. 51 runs for the new algorithm, LSMOP7;
20. 51 runs for gamultiobj, LSMOP7;
21. 51 runs for sp-MODE II, LSMOP7;
22. 51 runs for the new algorithm, LSMOP8;
23. 51 runs for gamultiobj, LSMOP8;
24. 51 runs for sp-MODE II, LSMOP8;
25. 51 runs for the new algorithm, LSMOP9;
26. 51 runs for gamultiobj, LSMOP9;
27. 51 runs for sp-MODE II, LSMOP9.

The poultry dataset will not be used. This dataset, from Monteiro and Reynoso-Meza (2017) and shown in the 1.6.10, is under re-evaluation by the company in order toensure it is more detailed and accurate to the current business needs.

Besides, for all the algorithms the following stopping criteria were considered:

- Maximum number of generations allowed: $200 * NumberOf\,V ariables$ (applicable for the three algorithms);
- Mutation α probability: 0.5 (applicable for the new algorithm and gamultiobj);
- Mutation x Recombination probability: 0.8 (applicable for the new algorithm and gamultiobj);
- Initial population size: 20 (applicable for the three algorithms);
- Maximum front size per front level: 40 (applicable for the new algorithm);
- Maximum number of generations in sequence without improvements before stopping the algorithm: 5 (applicable for the new algorithm and gamultiobj);

- Percentage used to determine if a generation was improved in relation to the previousone: 0.0001 (applicable for the new algorithm and gamultiobj);
- Maximum non-domination level to be considered between generations: 5 (applicable for the new algorithm);
- Maximum non-domination level to be considered for the local search: 1 (applicable for the new algorithm);
- Number of random neighbors for the local search: 50 (applicable for the new algorithm);
- Limit in seconds: 300 (applicable for the three algorithms);
- Flag responsible to whether the algorithm should forcefully prune each generation: *false* for the problems LSMOP5 and LSMOP8, *true* for all the others (applicable for the new algorithm).

4.2 Pareto fronts and Hypervolume

The hypervolume determination algorithm, as implemented in MATHWORKS (2015), starts by first defining the utopia and nadir points out of the consolidated Pareto front approximation determined for that problem and for each algorithm. For instance, for a given problem *n*, 51 runs for the sp-MODE II, 51 runs for the gamultiobj and another 51 runs for the new algorithm were executed. Each one of these 153 runs had their own Pareto front approximations formed by different non-domination levels apart from some cases as explained later in more detail. Regardless, those 153 different fronts were summarized into three fronts (one per algorithm) and considering only the non-dominated solutions (i.e. the first non-domination level) under that algorithm. As such, the resulting Pareto front approximation charts show three fronts, each for each algorithm.

All of the solution values found for these fronts are normalized considering the fronts generated for all the algorithms – consequently, the utopia point has the value 0 assigned for all of its objectives and the nadir point has 1 assigned for all of its objectives. Considering these two points, a high amount (in these tests, 1000000) of random solutions (henceforth named *samples*) is generated between these two points and across the hyperplane (i.e. for all the objectives).

Then, for each one of the solutions belonging to the front of a given algorithm, only the samples fully dominated by the solution are considered. These samples are included in a counter and are also removed from the list (so that they are not tested another time against another solution). The overall count is then divided by the original number. As such, if in a given case 700000 samples were found to be dominated out of 1000000,

then the hypervolume calculation results in 0.700 (700000/1000000). Therefore, if all the samples are found to be dominated by the solutions in the front the hypervolume will be 1.000. On the other hand, if no solutions dominated the samples the hypervolume will be 0.000. By conclusion, the closest the algorithm is to 1.000 the better it will be from a Pareto front approximation analysis standpoint.

4.3 Findings

Initially, with few (1000) variables the new algorithm took longer and was not able to provide the best results - although it fared generally better than the sp-MODE II algorithm, it stood behind gamultiobj. Also, it only had its execution completed as soon as it reached the time limit. On the other hand, in more than half of the problems it found at least one solution that fared better in at least one objective than all the others found by the other algorithms as seen in the Figure 15.

Moreover, as soon as the number of variables was increased it was possible to better see the benefits of the new algorithm. When it reached 15000 variables it was able to generate a greater amount of solutions (seen in the Figure 17) using way less resources than gamultiobj, as shown in the Figure 25. sp-MODE II, for instance, was not able to be initiated since it required more than 70 GB of available RAM for problems with this size. The same situation happened with 30000 and 50000 variables, but with gamultiobj.

Additionally, it is important to note the algorithm, as shown in the Table 4, could properly finish itself earlier even with a high amount of variables (116 seconds for LSMOP2 with 50000 variables) since it quickly found it was not able to improve itself. As also shown in the Table 5, the solutions found by it, even with a high amount of variables, were diverse and could even be better improved if more time was allocated for its execution. This is noted by the differences between its minimum, maximum and median hypervolumes for 30000 and 50000 variables. If the three values were consistent one could conclude the solutions found were very similar among themselves for all the runs and, therefore, the algorithm would hardly find better solutions with more time.

This is further corroborated by the Figures 20 to 24. In these figures, it is possible to verify that, in general, as the problems grew in complexity, the new algorithm kept providing more consistent values when compared against other algorithms. Starting in the Figure 21 and being more noticeable in the Figure 22, the new algorithm has a thicker violin area as well as less outliers. Contrarily, the gamultiobj box plot has a larger areawith a less noticeable violin plot and with more outliers.

Supporting the hypervolumes shown in the Table 5, the Figures 15 to 19 also show the behavior of the Pareto front approximation as the algorithm and the problem complexity were changed. In these figures, it is better noticeable that the Pareto front

approximation generated by the new algorithm already have some solutions that are members of the overall front (i.e. when the three algorithms are considered) with 1000 variables. Then, as the number of variables grow, more solutions found by the new algorithm are closer and/or part of the overall front until the point where only the new algorithm is capable of generating a Pareto front approximation at all.

The Table 6 also shed a different view on the hypervolumes. This table uses the Wilcoxon signed rank test as shown by Devore (2011). The objective of this test is to evaluate the *null hypothesis* (i.e. whether there is no significant difference between both populations). If the null hypothesis is *rejected*, then both sets are considered to be different between themselves. Considering the results from the 51 runs for sp-MODE II and the 51 runs for gamultiobj compared against the 51 runs for the new algorithm for each problem, it is possible to verify that the null hypothesis was rejected against all problems and all sizes - therefore, the results by the new algorithm are statistically different from the ones presented by the other algorithms for all the problems.

From another perspective, the Figure 25 shows the algorithm comparison concerning specifically the resource usage. From a performance standpoint, the memory and footprint recorded consumption were consistent for the new algorithm. Considering the memory and usage patterns were similar for all the algorithms for a given problem and size combination, the Figure 25 to use the results found for LSMOP5, where all the 51 runs take place sequentially for each one of the algorithms. From a processor usage standpoint, the metric used was the % Privileged Time as used in Gough, Steiner and Saunders (2015) and, for memory, the number of Particular Bytes was used, following Darabont, Kiss and Domokos (2015). On both cases, the best-case scenario is to have both values as small as possible,representing lower resource usage.

For better comparison purposes, the *y*-axis scale remains unchanged for all the scenarios. As seen in the Figure 25, the memory recorded usages are similar between the three algorithms as far as 1000 variables are concerned, and with around 1 GB used. The sp-MODE II algorithm recorded faster times and lower processor usage while gamultiobj, although with the best hypervolumes, used way more processor resources. Then, with 5000 variables, sp-MODE II used way more memory resources with around 4 to 5 GB in memory usage while at the same time keeping the processor usage low. On the other hand, the new algorithm used the least memory from them all, with less than 2 GB used. Comparing the changes from the problems with this magnitude with the ones with 1000 variables it is possible to notice the other algorithms started to use way more resources as the problem became more complex while the new algorithm kept itself using less resources.

With 15000 variables, sp-MODE II was unable to run due to an out-of-memory error since it attempted to allocate more than 70 GB. The new algorithm kept with its premise of low resource usage, consuming around 2 GB in memory while gamultiobj

used

5 times more than that. This behavior also made gamultiobj cause an out-of-memory error with the problems with 30000 and 50000 variables. For both cases it is visible that the new algorithm kept its low memory footprint, still consuming less than 2 GB in memory while using less than 40% of the processor resources.

With 30000 variables, both gamultiobj and sp-MODE II were unable to generate any solutions. Therefore, for all the problems with this size as well as 50000 variables, the new algorithm was the only one capable of generating the solutions.

It is also important to highlight the scalability of the proposed algorithm. When the memory and CPU consumption are considered it is possible to note the low footprint given by the new algorithm. In practice, if the algorithm was executed in a corporate environment in an off-the-shelf computer, its user could arguably keep working in another tasks in parallel using the same equipment without the need to allocate a separate server. Moreover, it also adds the possibility for each analyst to run its own algorithm without the need to worry about the server shared usage, for example.

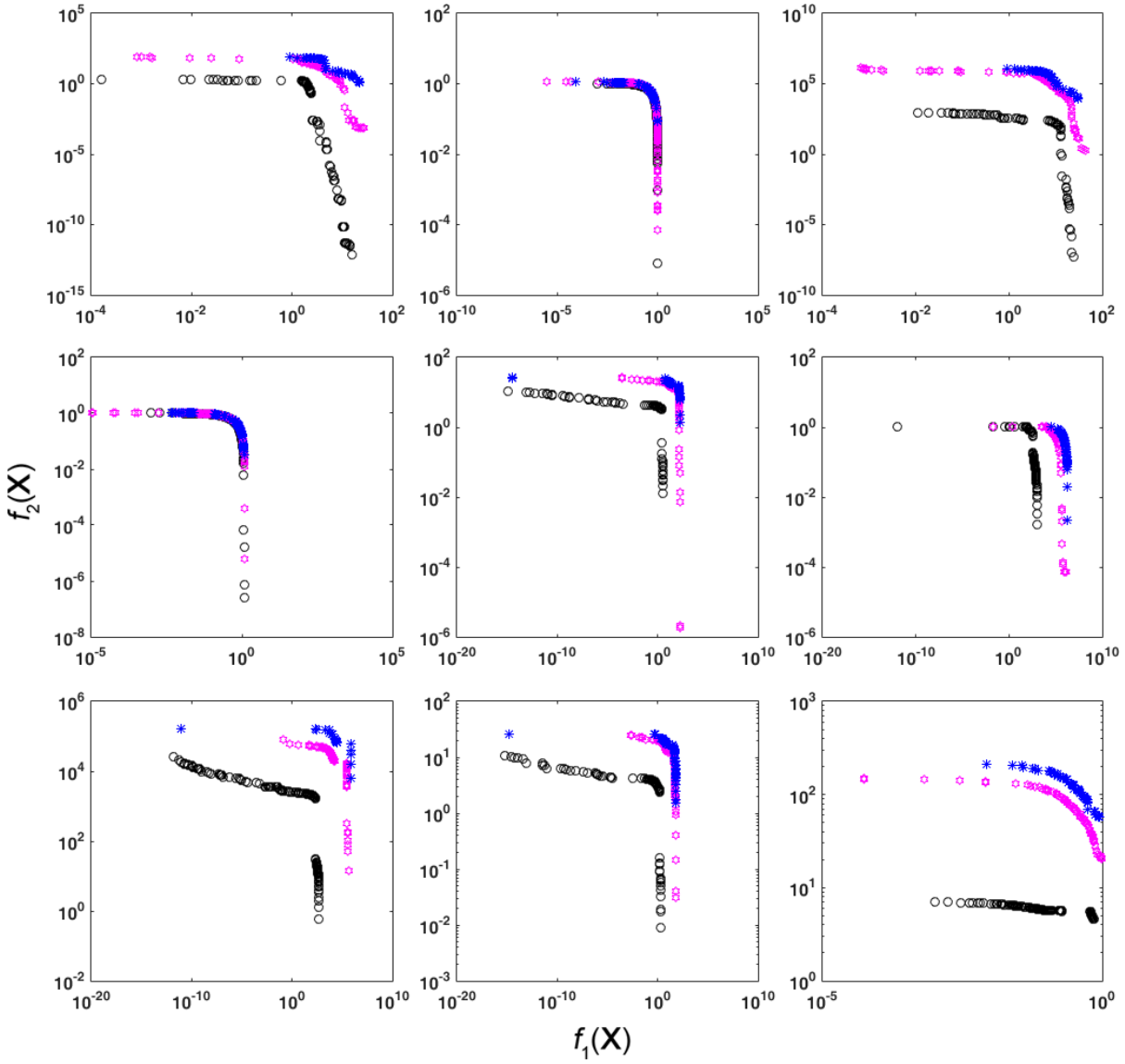

Figure 15 – The results for all the problems tested with 1000 variables. From left to right:first row - LSMOP1, LSMOP2, LSMOP3. Second row: LSMOP4, LSMOP5, LSMOP6. Third row: LSMOP7, LSMOP8, LSMOP9. Circles: gamultiobj, asterisks: sp-MODE II, stars: new algorithm.

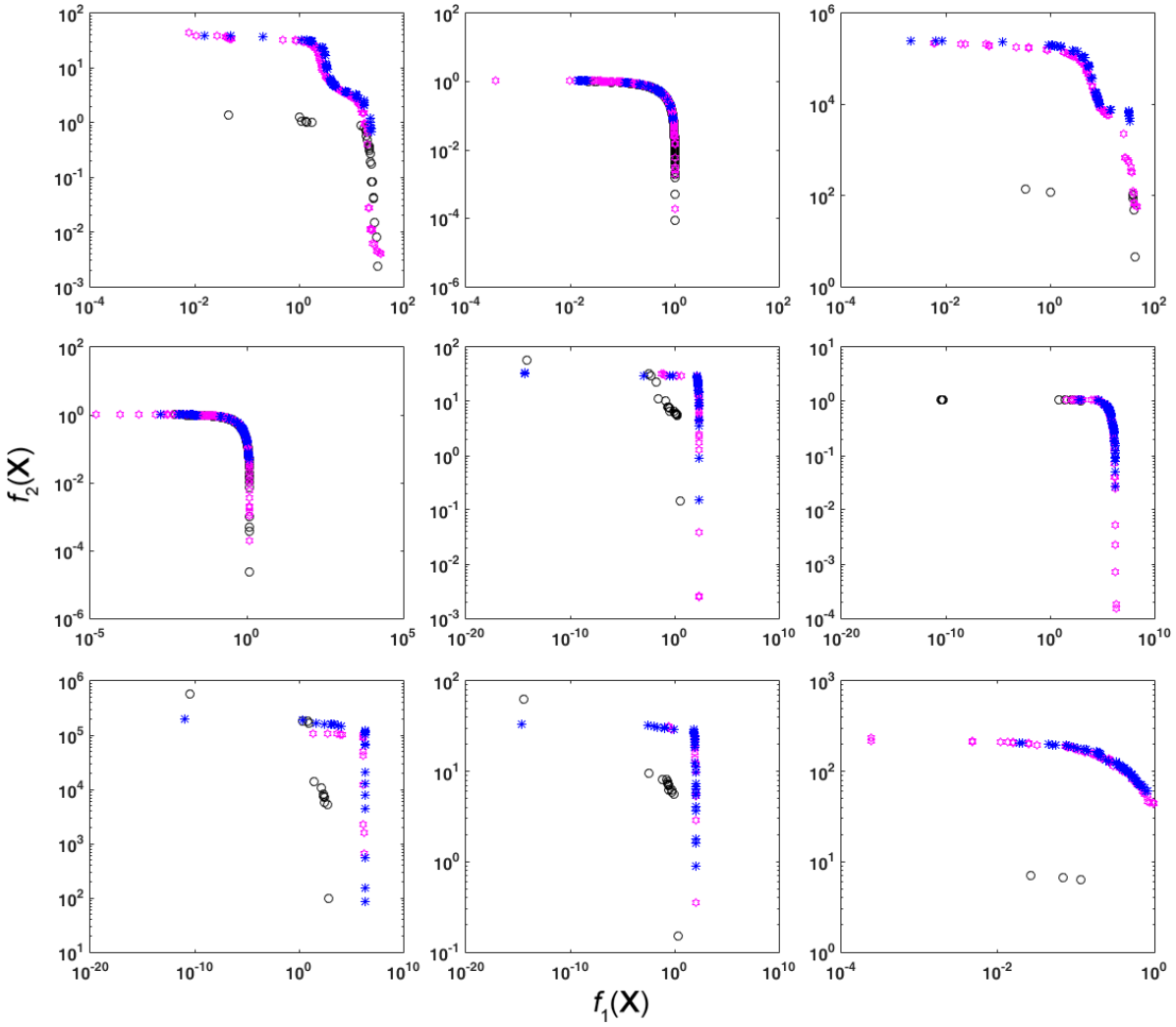

Figure 16 – The results for all the problems tested with 5000 variables. From left to right: first row - LSMOP1, LSMOP2, LSMOP3. Second row: LSMOP4, LSMOP5, LSMOP6. Third row: LSMOP7, LSMOP8, LSMOP9. Circles: gamultiobj, asterisks: sp-MODE II, stars: new algorithm.

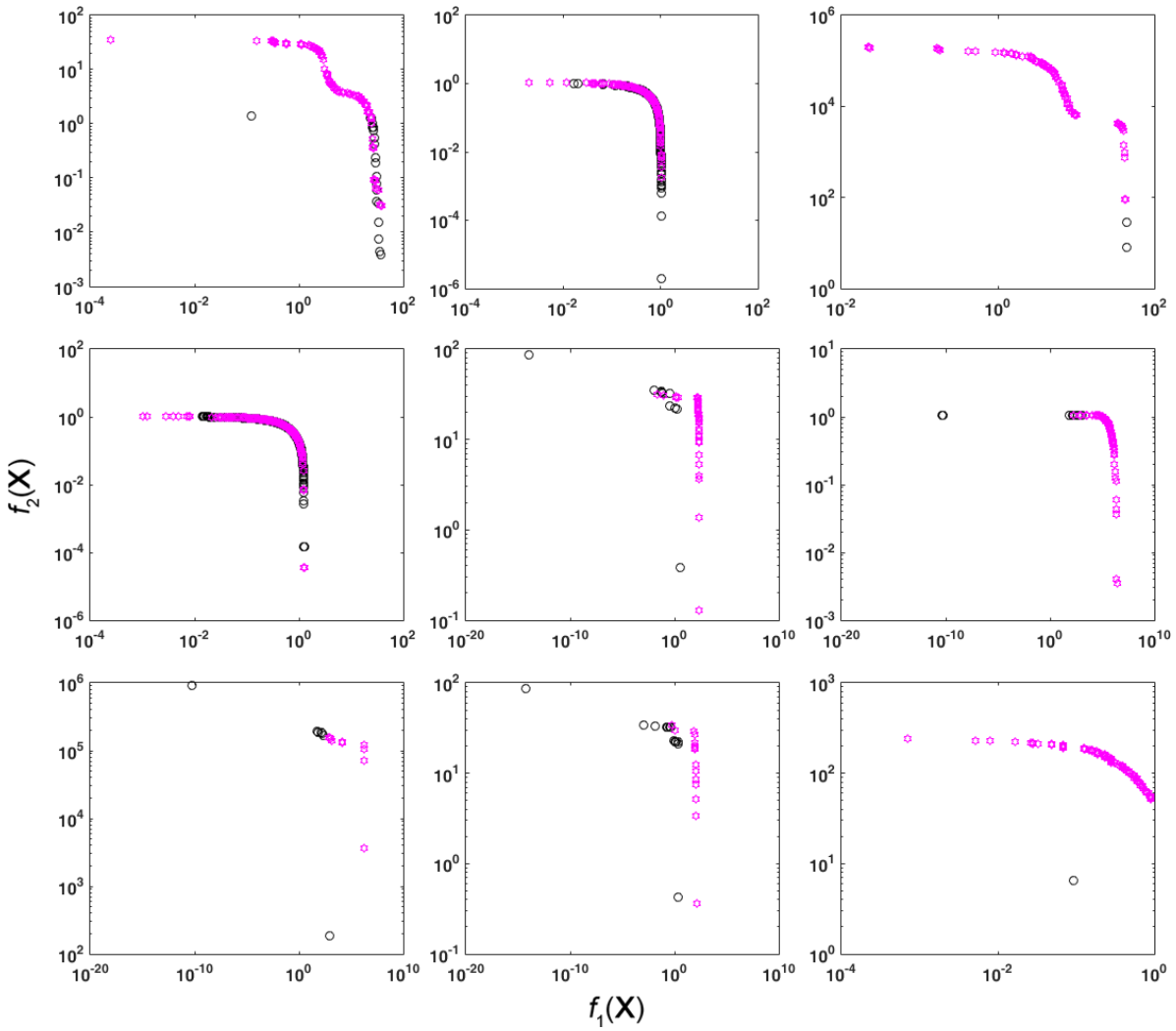

Figure 17 – The results for all the problems tested with 15000 variables. From left to right:first row - LSMOP1, LSMOP2, LSMOP3. Second row: LSMOP4, LSMOP5, LSMOP6. Third row: LSMOP7, LSMOP8, LSMOP9. Circles: gamultiobj, stars: new algorithm.

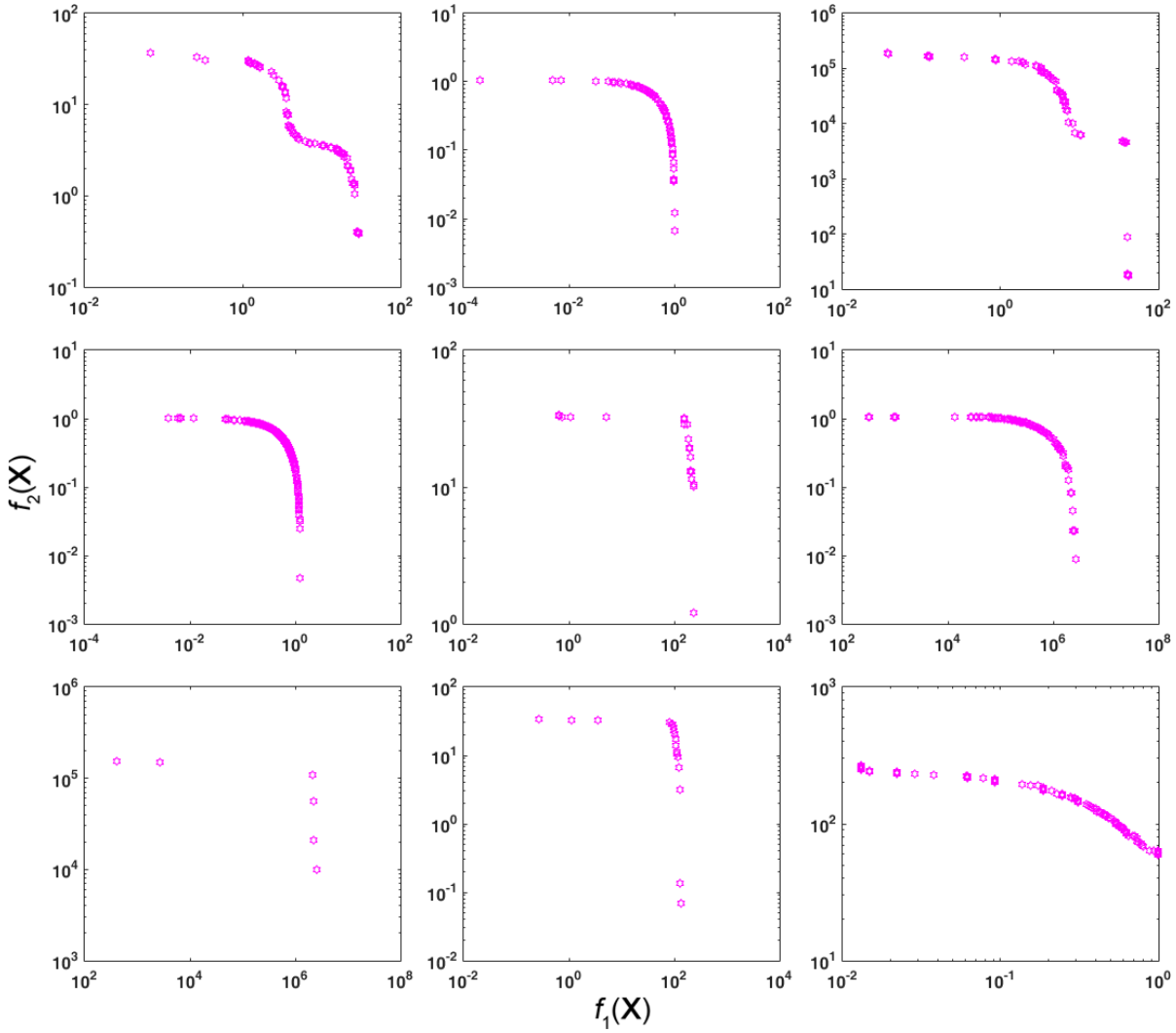

Figure 18 – The results for all the problems tested with 30000 variables. From left to right: first row - LSMOP1, LSMOP2, LSMOP3. Second row: LSMOP4, LSMOP5, LSMOP6. Third row: LSMOP7, LSMOP8, LSMOP9. Stars: new algorithm.

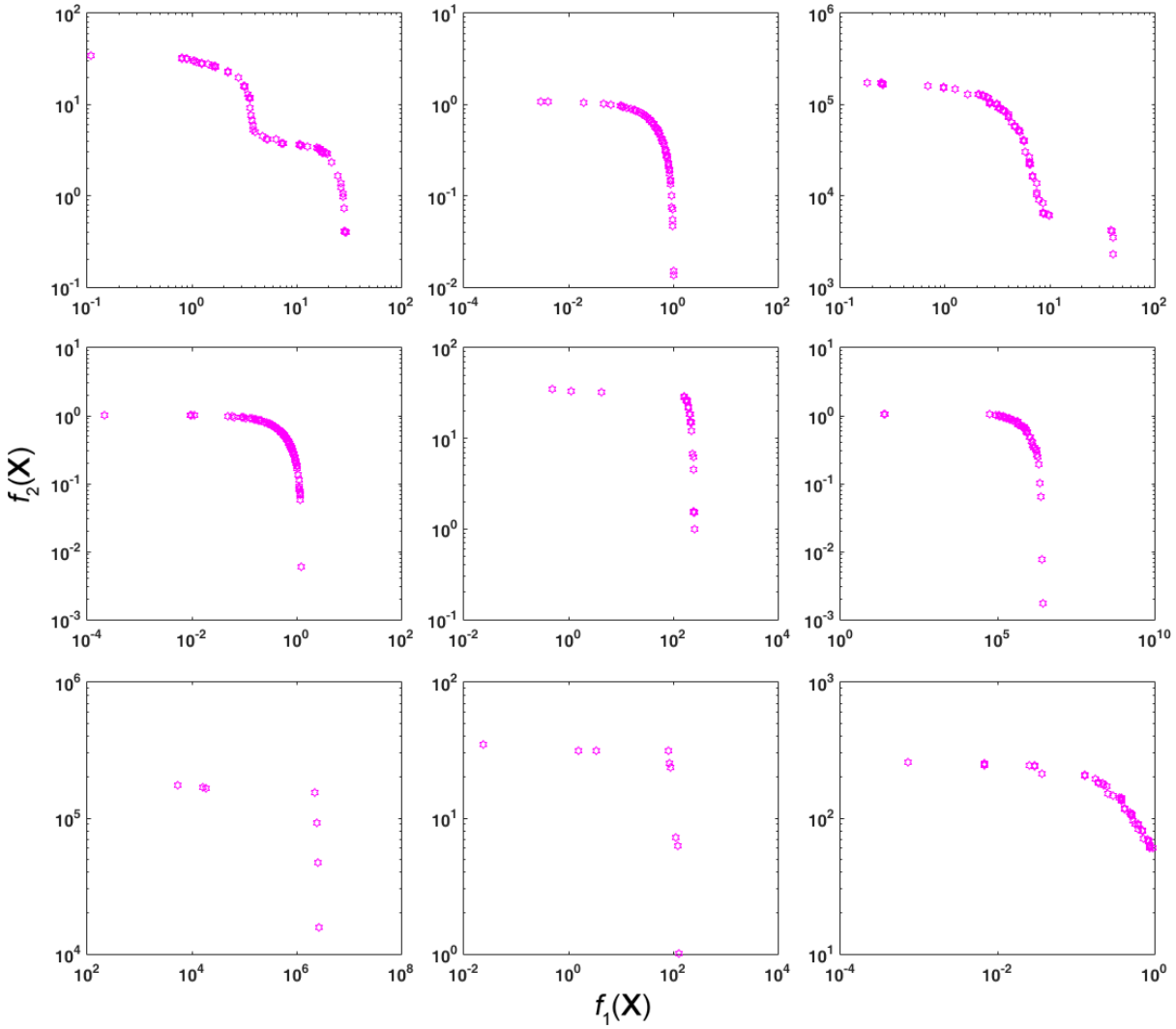

Figure 19 – The results for all the problems tested with 50000 variables. From left to right:first row - LSMOP1, LSMOP2, LSMOP3. Second row: LSMOP4, LSMOP5, LSMOP6. Third row: LSMOP7, LSMOP8, LSMOP9. Stars: new algorithm.

Problem	NVar	gamultiobj	sp-MODE II	new algorithm
LSMOP1	1000	204.6695	**18.8369**	300.5821
LSMOP2	1000	179.0439	77.5965	**56.4962**
LSMOP3	1000	194.1867	**22.2022**	300.6393
LSMOP4	1000	185.2759	**61.2616**	300.4007
LSMOP5	1000	203.9149	**17.5401**	302.7825
LSMOP6	1000	174.2944	**22.3451**	300.5714
LSMOP7	1000	207.0651	**22.9051**	300.5503
LSMOP8	1000	206.9397	**24.5861**	302.9819
LSMOP9	1000	203.9316	**23.1693**	300.8249
LSMOP1	5000	212.4453	**61.9577**	301.1190
LSMOP2	5000	212.4953	475.4843	**39.1407**
LSMOP3	5000	211.7594	**62.9883**	301.0384
LSMOP4	5000	**212.6598**	213.7962	301.1508
LSMOP5	5000	212.5400	**51.4895**	305.6372
LSMOP6	5000	210.5775	**67.8386**	301.3732
LSMOP7	5000	213.5732	**58.4579**	301.2659
LSMOP8	5000	213.5123	**65.1098**	304.5328
LSMOP9	5000	209.5527	**63.1830**	301.1826
LSMOP1	15000	320.0012	-	**301.9038**
LSMOP2	15000	318.4921	-	**44.1292**
LSMOP3	15000	326.5167	-	**301.2525**
LSMOP4	15000	317.6301	-	**302.4874**
LSMOP5	15000	316.6140	-	**305.6782**
LSMOP6	15000	315.4029	-	**302.1086**
LSMOP7	15000	316.7320	-	**301.6089**
LSMOP8	15000	317.1549	-	**312.3363**
LSMOP9	15000	313.5930	-	**301.2109**
LSMOP1	30000	-	-	**303.9869**
LSMOP2	30000	-	-	**71.1807**
LSMOP3	30000	-	-	**302.7238**
LSMOP4	30000	-	-	**305.0936**
LSMOP5	30000	-	-	**317.2387**
LSMOP6	30000	-	-	**301.6293**
LSMOP7	30000	-	-	**303.0287**
LSMOP8	30000	-	-	**315.4613**
LSMOP9	30000	-	-	**302.4290**
LSMOP1	50000	-	-	**302.9240**
LSMOP2	50000	-	-	**116.3987**
LSMOP3	50000	-	-	**304.2338**
LSMOP4	50000	-	-	**307.7581**
LSMOP5	50000	-	-	**313.8191**
LSMOP6	50000	-	-	**301.2501**
LSMOP7	50000	-	-	**304.3168**
LSMOP8	50000	-	-	**311.4810**
LSMOP9	50000	-	-	**302.5563**

Table 4 – Median time taken for each algorithm considering 51 runs. NVar represents the number of variables. The algorithms with the best median values are in bold.

Problem	NVar	gamultiobj			sp-MODE II			new algorithm		
		minimum	maximum	median	minimum	maximum	median	minimum	maximum	median
LSMO P1	100	**0.9897 06**	**0.9995 27**	**0.9989 33**	0.914 590	0.922 737	0.9169 51	0.9573 59	0.9721 75	0.966 355
LSMO P2	100	**0.5159 83**	**0.5369 64**	**0.5319 42**	0.467 864	0.473 460	0.4722 04	0.4882 40	0.5008 43	0.497 616
LSMO P3	100	**0.9865 51**	**0.9999 60**	**0.9998 81**	0.911 393	0.920 265	0.9163 40	0.9553 86	0.9666 92	0.960 246
LSMO P4	100	**0.5435 32**	**0.5715 30**	**0.5623 90**	0.491 091	0.496 510	0.4944 51	0.5130 55	0.5293 36	0.521 077
LSMO P5	100	**0.8929 86**	**0.9991 72**	**0.9974 82**	0.391 758	0.497 825	0.4274 28	0.6011 09	0.6670 64	0.628 847
LSMO P6	100	**0.9374 07**	**0.9998 19**	**0.9995 99**	0.607 114	0.644 470	0.6268 70	0.8719 86	0.9420 22	0.916 751
LSMO P7	100	**0.9893 53**	**0.9999 99**	**0.9996 44**	0.585 608	0.643 794	0.6159 92	0.9226 84	0.9639 04	0.947 478
LSMO P8	100	**0.8948 03**	**0.9985 73**	**0.9951 52**	0.493 677	0.533 760	0.5089 90	0.6724 84	0.7214 65	0.686 416
LSMO P9	100	**0.7271 11**	**0.9947 42**	**0.9910 79**	0.465 631	0.497 300	0.4834 26	0.6506 11	0.7749 93	0.695 966
LSMO P1	500	**0.9785 04**	**0.9937 91**	**0.9905 06**	0.554 203	0.577 637	0.5616 39	0.9556 49	0.9632 43	0.958 904
LSMO P2	500	**0.5043 13**	**0.5162 85**	**0.5128 29**	0.500 476	0.516 116	0.5074 50	0.4826 73	0.4932 89	0.488 354
LSMO P3	500	**0.9843 09**	**0.9995 39**	**0.9960 04**	0.926 591	0.939 028	0.9312 65	0.9463 54	0.9584 51	0.954 693
LSMO P4	500	**0.8839 99**	**0.9977 73**	**0.9952 79**	0.468 779	0.471 864	0.4703 81	0.9226 84	0.9639 04	0.947 478
LSMO P5	500	**0.9853 58**	**0.9999 76**	**0.9994 11**	0.921 576	0.933 001	0.9250 04	0.8548 19	0.8956 01	0.882 027
LSMO P6	500	**0.8852 46**	**0.9996 52**	**0.9471 14**	0.494 375	0.498 301	0.4968 39	0.6265 55	0.7260 76	0.677 455
LSMO P7	500	**0.7191 77**	**0.9967 53**	**0.9584 39**	0.554 203	0.577 637	0.5616 39	0.5257 13	0.6229 79	0.593 416
LSMO P8	500	**0.9488**	**0.9857**	**0.9797**	0.500 476	0.516 116	0.5074 50	0.9431	0.9515	0.945
LSMO P9	500									
LSMO P1	1000									
LSMO P2	1000									
LSMO P3	1000									
LSMO P4	1000									
LSMO P5	1000									
LSMO P6	1000									
LSMO P7	1000									
LSMO P8	1000									
LSMO P9	1000									

P5	00										
LSMO P6	300 00	**50**	**81**	**83**	-	-	-	82	07	874	
LSMO P7	300 00	**0.5115 24**	**0.5141 72**	**0.5133 71**	-	-	-	0.4731 44	0.4916 00	0.484 434	
LSMO P8	300 00	**0.9536 95**	**0.9995 76**	**0.9986 51**	-	-	-	0.9392 23	0.9497 65	0.941 039	
LSMO P9	300 00	**0.5329 45**	**0.5379 77**	**0.5358 47**	-	-	-	0.5023 06	0.5155 80	0.510 049	
LSMO P1	500 00	**0.7546 04**	**0.9933 60**	**0.9741 57**	-	-	-	0.6855 95	0.7013 02	0.688 771	
LSMO P2	500 00	**0.6577 02**	**0.9995 25**	**0.7717 05**	-	-	-	0.5714 92	0.6090 21	0.585 666	
LSMO P3	500 00	**0.8701 61**	**0.9997 15**	**0.9990 59**	-	-	-	0.8395 20	0.8599 87	0.845 907	
LSMO P4	500 00	**0.7481 59**	**0.9924 55**	**0.9892 29**	-	-	-	0.6737 50	0.6987 94	0.680 188	
LSMO P5	500 00	**0.5499 97**	**0.9982 70**	**0.9907 60**	-	-	-	0.4802 48	0.5622 70	0.540 451	
		-	-	-				**0.8380 55**	**0.8440 85**	**0.8403 06**	
		-	-	-				**0.4445 10**	**0.4761 45**	**0.4663 73**	
		-	-	-				**0.8846 19**	**0.8938 15**	**0.8858 42**	
		-	-	-				**0.4817 29**	**0.4920 74**	**0.4868 88**	
		-	-	-				**0.1562 84**	**0.2152 70**	**0.1922 26**	
		-	-	-				**0.5641 66**	**0.6149 24**	**0.6042 63**	
		-	-	-				**0.0982 36**	**0.2515 62**	**0.2232 34**	
		-	-	-				**0.1705 07**	**0.2025 35**	**0.1840 88**	

	- - -		**0.660507**	**0.675692**	**0.668866**
	- - -		**0.830194**	**0.839140**	**0.831989**
	- - -		**0.467906**	**0.481969**	**0.476713**
			0.885075	**0.891633**	**0.887729**
			0.478048	**0.491921**	**0.485269**
			0.209841	**0.229832**	**0.220266**
			0.568698	**0.579356**	**0.573581**
			0.054604	**0.111607**	**0.081000**
			0.196041	**0.227283**	**0.208853**
			0.618596	**0.683585**	**0.669033**

Table 5 – Hypervolumes for each algorithm considering 51 runs. NVar represents the number of variables. The algorithms with the best median values are shown in bold.

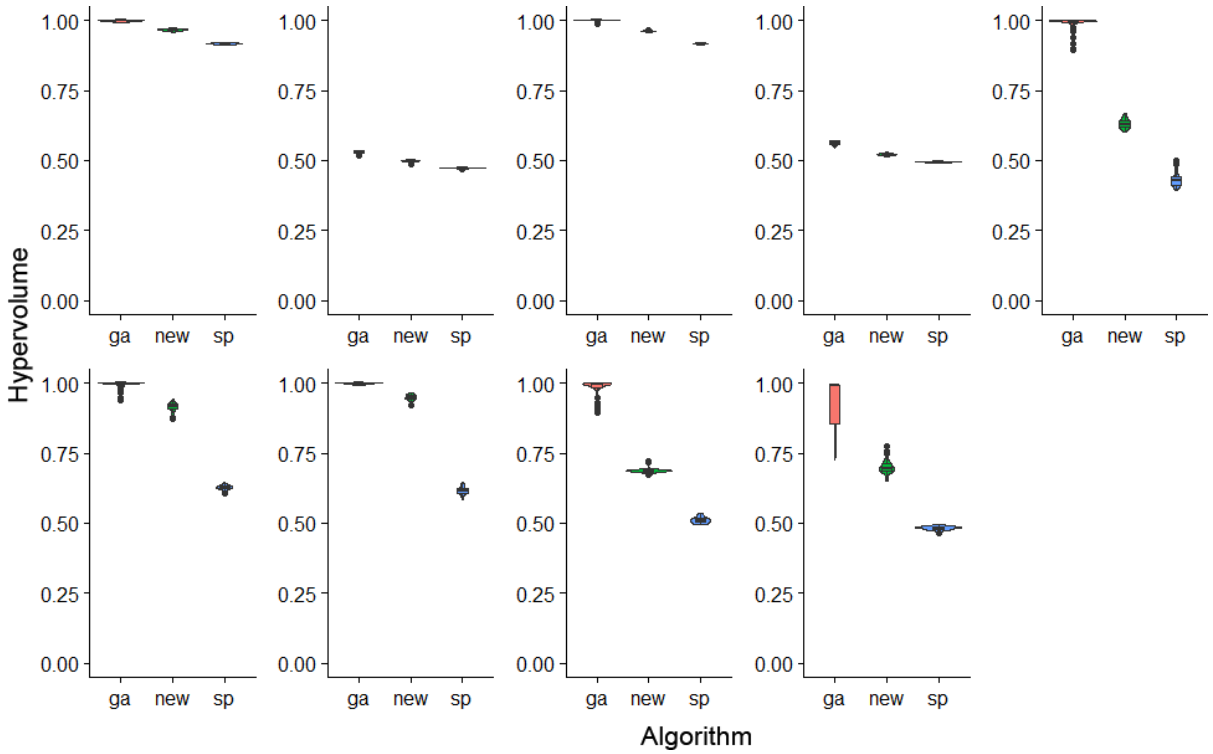

Figure 20 – The hypervolume distribution for all the problems tested with 1000 variables. The vertical axis represents the hypervolume values, where *ga* represents the runs with gamultiobj, *sp* with sp-MODE II and *new* with the new algorithm. The density of the violin refers to the distribution of all the hypervolume values for a given problem, the box height determines the interquartile range, the strip in the box represents the median and the dots represent the outliers.

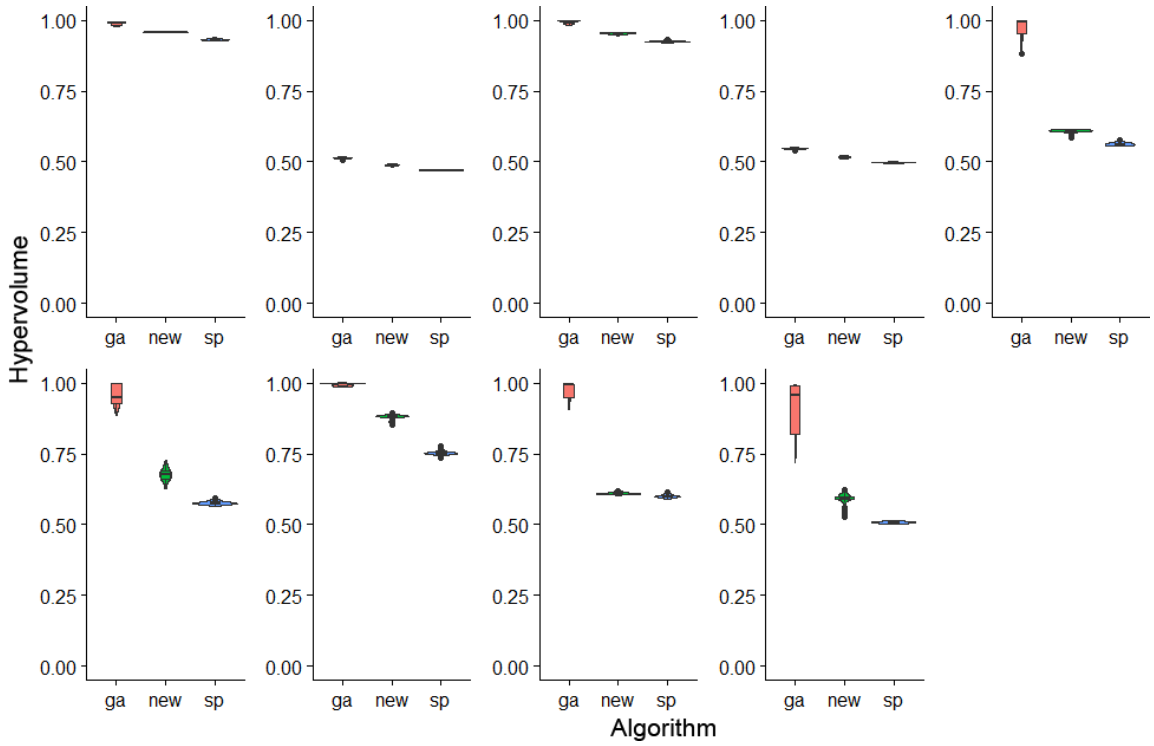

Figure 21 – The hypervolume distribution for all the problems tested with 5000 variables. The vertical axis represents the hypervolume values, where *ga* represents the runs with gamultiobj, *sp* with sp-MODE II and *new* with the new algorithm. The density of the violin refers to the distribution of all the hypervolume values for a given problem, the box height determines the interquartile range, the strip in the box represents the median and the dots represent the outliers.

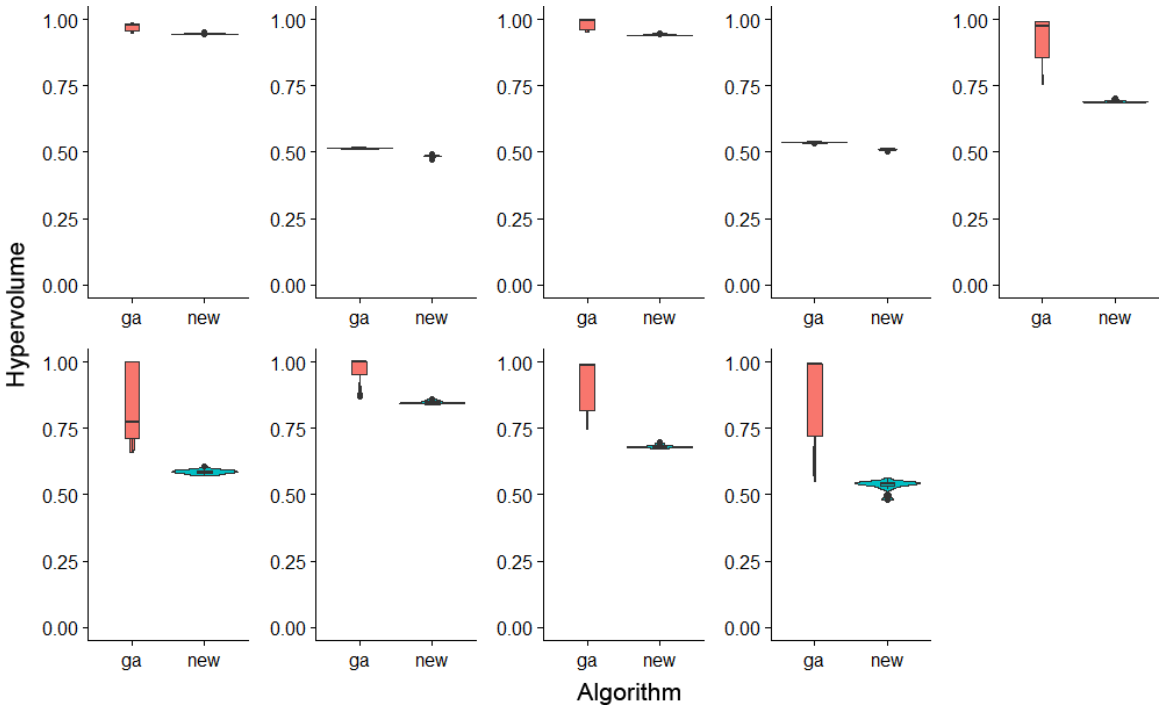

Figure 22 – The hypervolume distribution for all the problems tested with 15000 variables. The vertical axis represents the hypervolume values, where *ga* represents the runs with gamultiobj and *new* with the new algorithm. The density of the violin refers to the distribution of all the hypervolume values for a given problem, the box height determines the interquartile range, the strip in the box represents the median and the dots represent the outliers.

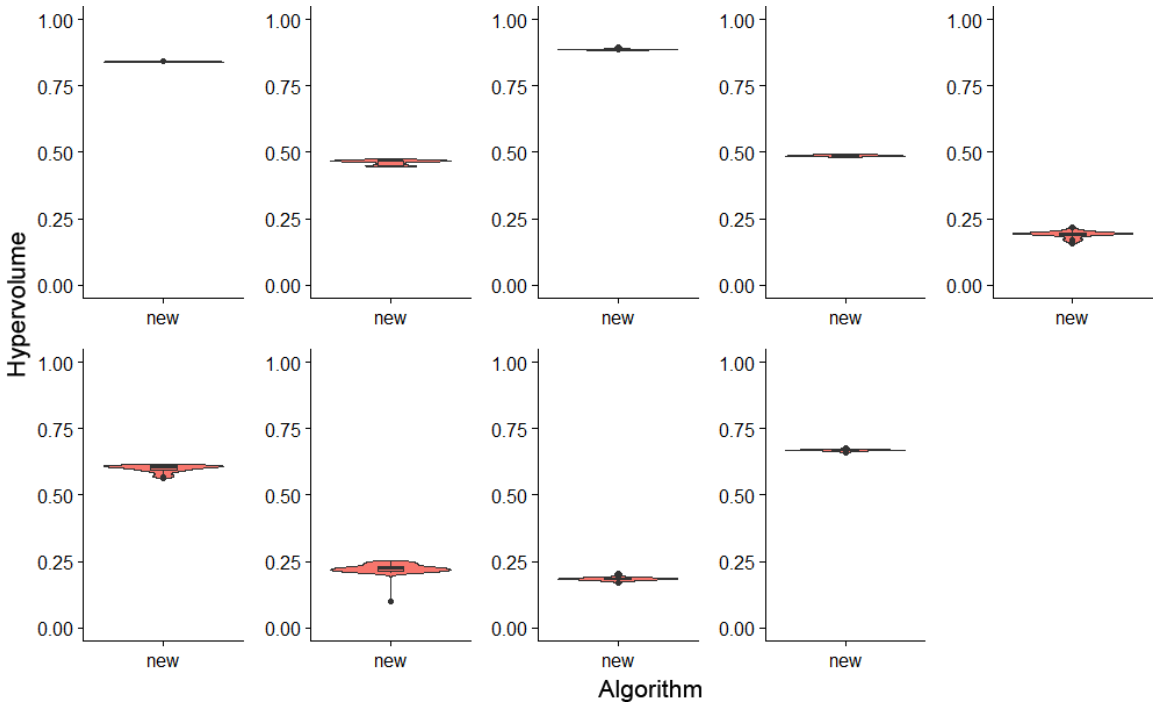

Figure 23 – The hypervolume distribution for all the problems tested with 30000 variables. The vertical axis represents the hypervolume values, where *new* represents the runs with the new algorithm. The density of the violin refers to the distribution of all the hypervolume values for a given problem, the box height determines the interquartile range, the strip in the box represents the median and the dots represent the outliers.

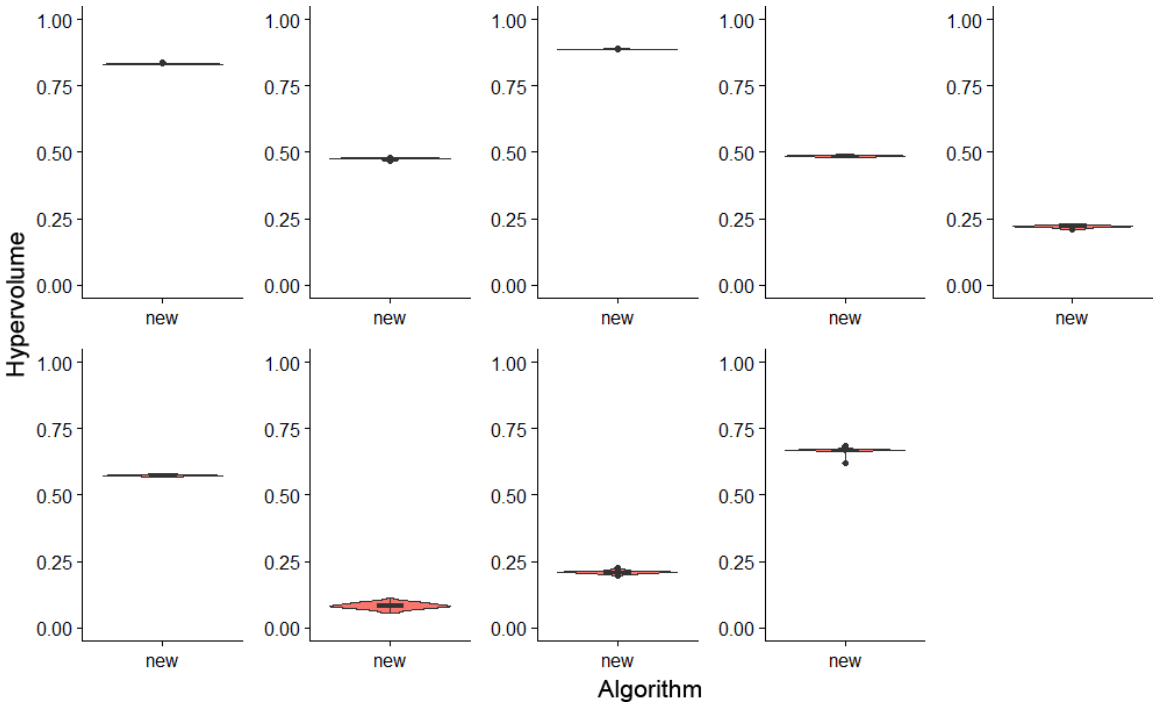

Figure 24 – The hypervolume distribution for all the problems tested with 50000 variables. The vertical axis represents the hypervolume values, where *new* represents the runs with the new algorithm. The density of the violin refers to the distribution of all the hypervolume values for a given problem, the box height determines the interquartile range, the strip in the box represents the median and the dots represent the outliers.

Problem	NVar	new vs. gamultiobj	new vs sp-MODE II
LSMOP1	1000	true	true
LSMOP2	1000	true	true
LSMOP3	1000	true	true
LSMOP4	1000	true	true
LSMOP5	1000	true	true
LSMOP6	1000	true	true
LSMOP7	1000	true	true
LSMOP8	1000	true	true
LSMOP9	1000	true	true
LSMOP1	5000	true	true
LSMOP2	5000	true	true
LSMOP3	5000	true	true
LSMOP4	5000	true	true
LSMOP5	5000	true	true
LSMOP6	5000	true	true
LSMOP7	5000	true	true
LSMOP8	5000	true	true
LSMOP9	5000	true	true
LSMOP1	15000	true	-
LSMOP2	15000	true	-
LSMOP3	15000	true	-
LSMOP4	15000	true	-
LSMOP5	15000	true	-
LSMOP6	15000	true	-
LSMOP7	15000	true	-
LSMOP8	15000	true	-
LSMOP9	15000	true	-

Table 6 – Wilcoxon signed rank test results compared against the median hypervolumes found for each algorithm and problem. *True* indicates a rejection of the null hypothesis at 5% significance level while *false* indicates a failure to reject the null hypothesis. The problems with 30000 and 50000 variables are not availablein this table since only the new algorithm was capable to run them.

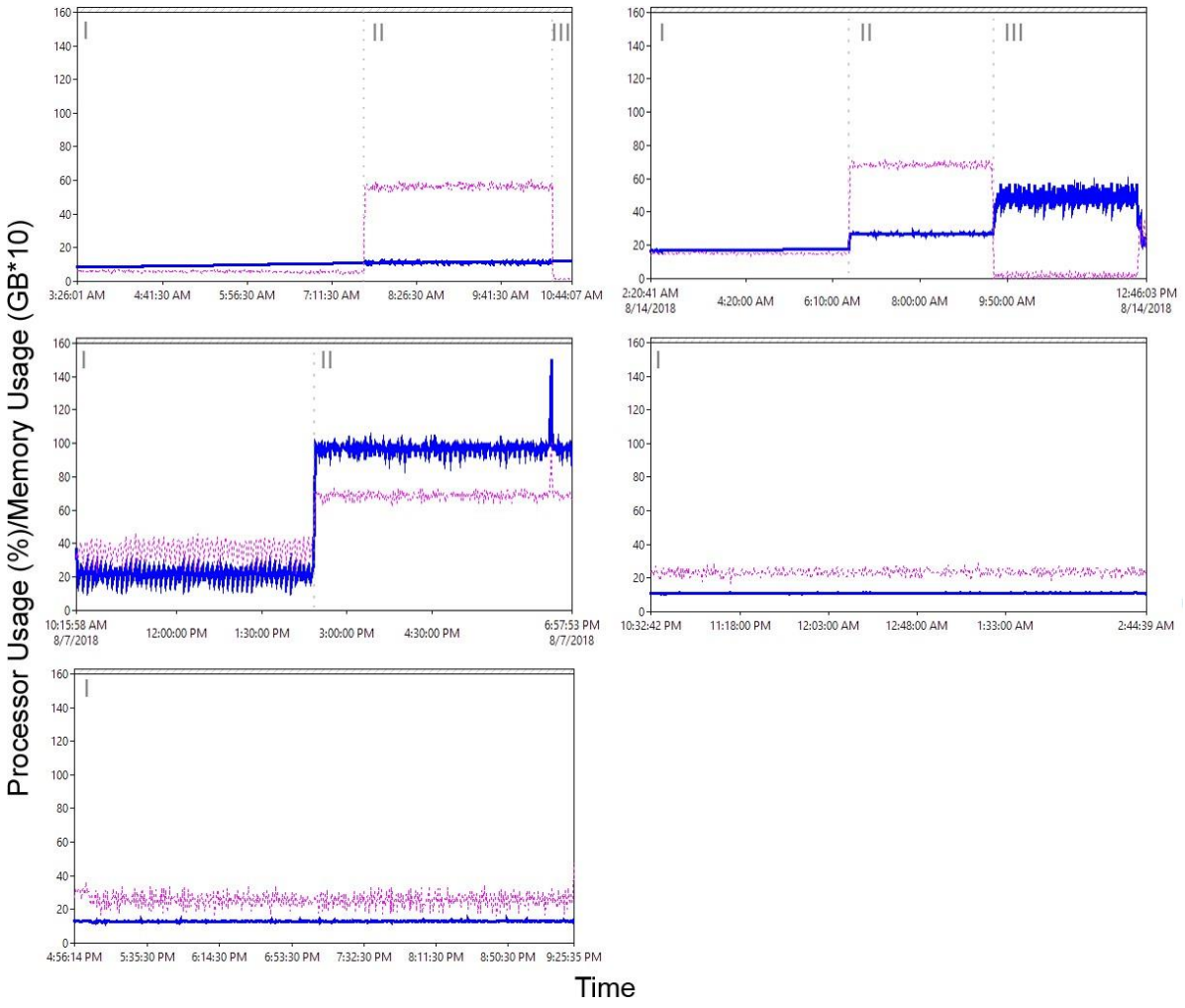

Figure 25 – The memory and processor usages registered through the perfmon tool. From top to bottom, left to right: results for LSMOP5 with 1000 variables, 5000 variables, 15000 variables, 30000 and 50000 variables. The memory usage is registered by the continuous lines while the processor usage is registered by the dashed lines. The *y*-axis represents the percentage of processor usage, for the processor usage lines (meaning the value of 60 represents 60% in processor usage), and the usage in hundreds of megabytes in memory for the memory usage lines (meaning the value of 60 means 6.0 GB in memory usage). The vertical, dotted lines represent the division between the algorithm execution as referred in the Section 4.1 (I: new algorithm; II: gamultiobj; III: sp-MODE II).

5 CONCLUSIONS

The key objective of this research was to present a new strategy targeted to multi-objective problems with a large amount of variables. This strategy must focus first on the resource usage with the trade-off of possibly producing solutions that would perform worse than others generated by other strategies. On this end, the suggested strategy,
implemented in MATLAB® proved it could be easily scalable offering low resource usage to the point of being capable to run problems with more than 50000 variables in off-the-shelf computers with less memory available. Moreover, this algorithm was also able to offer better performance as soon as the problems grew in complexity. From a solution quality standpoint, the solutions found were mildly inferior than the solutions generated by the best algorithm. However, it is equally important to note these scenarios happened only when the problem sizes were way smaller - at the same time, in the scenarios were the proposed algorithm is intended to run better not only it used few computational resources, but was the only one to effectively generate new solutions and their offspring. It is important to note that all of the key objectives originally expected by this strategy were met successfully within the scope of the problems used in the test.

Considering these criteria, while the findings analyzed were satisfactory, more opportunities are now available. For example, while the algorithm worked as intended in MATLAB®, an open possibility is to port the algorithm to the R and Python programming environments. Furthermore, with the user experience and usability in mind, a visual front-end can be created for the problem designers. Although the original idea is to ensure the algorithm can be executed in an off-the-shelf computer, it is also possible to use it in a dedicated server in order to reduce the time taken for a given run.

Another possibility is to further improve the proposed algorithm or offer an alternative to the same problem. Other improvements could include, for example, the deployment of other local search and offspring generation algorithms as either additional, sequential steps or parallel ones. An alternative, on the other hand, could make use of market data in order to help the business decisions *before* the production plan itself - i.e. in determining the demand with greater accuracy. This implies in problems with even greater magnitude that, when combined with the production plan itself, can further increase the expected profits as well as reducing possible issues caused by the production planning (i.e. additional costs with stock management due to material that could not be sold due to changes in the market demand).

However, these scenarios represent a harder challenge that, although it is indeed feasible, might require additional time. Therefore, these possibilities - summed with the

poultry database fixes from the business lines - can be implemented and tested in later opportunities.

Finally, this research also enables future steps such as the inclusion of optimizations to run in servers and/or clusters; the use of additional algorithms to pre-process the multi-objective problem before executing it; and the use of surrogate models and/or sensitivity analysis.

References

ABDULLAH, L.; ADAWIYAH, C. R. Simple additive weighting methods of multi criteria decision making and applications: A decade review. *International Journal of Information Processing and Management*, Advanced Institutes of Convergence Information Technology,
v. 5, n. 1, p. 39, 2014. Cited in the page 34.

AHN, C. W. et al. A hybrid multiobjective evolutionary algorithm: Striking a balance with local search. *Mathematical and Computer Modelling*, Elsevier, v. 52, n. 11-12, p. 2048–2059, 2010. Cited in the page 48.

AKHTAR, T.; SHOEMAKER, C. A. Multi objective optimization of computationally expensive multi-modal functions with rbf surrogates and multi-rule selection. *Journal of Global Optimization*, Springer, v. 64, n. 1, p. 17–32, 2016. Cited in the page 45.

ALBLIWI, S. et al. Critical failure factors of lean six sigma: a systematic literature review. *International Journal of Quality & Reliability Management*, Emerald Group Publishing Limited, v. 31, n. 9, p. 1012–1030, 2014. Cited in the page 21.

ANTONIO, L. M.; COELLO, C. A. C. Use of cooperative coevolution for solving large scale multiobjective optimization problems. In: IEEE. *2013 IEEE Congress on Evolutionary Computation (CEC)*. Cancun, Mexico: IEEE, 2013. p. 2758–2765. Cited 2times in the pages 34 and 47.

BEHZADIAN, M. et al. Promethee: A comprehensive literature review on methodologies and applications. *European Journal of Operational Research*, Elsevier, v. 200, n. 1, p. 198–215, 2010. Cited 2 times in the pages 35 and 36.

BERGH, F. Van den; ENGELBRECHT, A. P. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 8, n. 3, p. 225–239, 2004. Cited 3 times in the pages 45, 46, and 47.

BEYER, H.-G.; SENDHOFF, B. Robust optimization–a comprehensive survey. *ComputerMethods in Applied Mechanics and Engineering*, Elsevier, v. 196, n. 33, p. 3190–3218, 2007. Cited in the page 40.

BLASCO, X. et al. A new graphical visualization of n-dimensional pareto front for decision-making in multiobjective optimization. *Information Sciences*, Elsevier, v. 178,n. 20, p. 3908–3924, 2008. Cited in the page 40.

BONISSONE, P. P.; SUBBU, R.; LIZZI, J. Multicriteria decision making (mcdm): a framework for research and applications. *IEEE Computational Intelligence Magazine*, IEEE, v. 4, n. 3, p. 48–61, 2009. Cited in the page 33.

BONYADI, M. R. et al. Evolutionary computation for multicomponent problems: opportunities and future directions. *arXiv preprint arXiv:1606.06818*, 2016. Cited in the page 41.

BOUSSAÏD, I.; LEPAGNOT, J.; SIARRY, P. A survey on optimization metaheuristics. *Information Sciences*, Elsevier, v. 237, p. 82–117, 2013. Cited in the page 22.

CAI, L. et al. A clustering-ranking method for many-objective optimization. *Applied Soft Computing*, Elsevier, v. 35, p. 681–694, 2015. Cited in the page 43.

CHAUDHURI, K.; DASGUPTA, S. Rates of convergence for nearest neighbor classification. In: *Advances in Neural Information Processing Systems*. Montreal, Canada: NIPS, 2014. p.3437–3445. Cited in the page 78.

CHENG, R. et al. Test problems for large-scale multiobjective and many-objective optimization. *IEEE Transactions on Cybernetics*, IEEE, 2016. Cited 4 times in the pages23, 45, 49, and 50.

CHRISTOPHER, M. *Logistics & supply chain management*. Harlow, United Kingdom: Pearson UK, 2016. Cited in the page 58.

COELLO, C. A. C. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, Springer, v. 1, n. 3, p. 269–308, 1999. Cited in the page 33.

COELLO, C. A. C. et al. *Evolutionary algorithms for solving multi-objective problems*.New York, U.S.A.: Springer New York, 2007. Cited 2 times in the pages 40 and 67.

COELLO, C. C. Evolutionary multi-objective optimization: a historical view of the field. *IEEE Computational Intelligence Magazine*, IEEE, U.S.A., v. 1, n. 1, p. 28–36, 2006. Cited in the page 33.

CORNE, D. W.; KNOWLES, J. D. Techniques for highly multiobjective optimisation: some nondominated points are better than others. In: ACM. *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. London, United Kingdom:ACM Press, 2007. p. 773–780. Cited in the page 40.

CRAFT, D. et al. Shared data for intensity modulated radiation therapy (imrt) optimization research: the cort dataset. *GigaScience*, BioMed Central, v. 3, n. 1, p. 37,2014. Cited in the page 22.

DARABONT, Ö.; KISS, K. J.; DOMOKOS, J. Performance analysis of remote desktop virtualization based on hyper-v versus remote desktop services. *MACRo 2015*, De Gruyter Open, v. 1, n. 1, p. 125–134, 2015. Cited in the page 86.

DAS, S. et al. Real-parameter evolutionary multimodal optimization - a survey of the state-of-the-art. *Swarm and Evolutionary Computation*, Elsevier, v. 1, n. 2, p. 71–88, 2011.Cited in the page 40.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 6, n. 2, p. 182–197, 2002. Cited inthe page 48.

DEB, K. et al. Scalable test problems for evolutionary multiobjective optimization. Springer, p. 105–145, 2005. Cited in the page 50.

DEVORE, J. L. *Probability and Statistics for Engineering and the Sciences*. U.S.A.:Cengage learning, 2011. Cited in the page 86.

FARINA, M.; DEB, K.; AMATO, P. Dynamic multiobjective optimization problems: test cases, approximations, and applications. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 8, n. 5, p. 425–442, 2004. Cited 2 times in the pages 40 and 44.

FLANDERS, F.; GILLESPIE, J. R. *Modern livestock & poultry production*. U.S.A.: Cengage Learning, 2015. Cited in the page 59.

FONSECA, C. M.; FLEMING, P. J. et al. Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. In: *Proceedings of the 5th International Conference on Genetic Algorithms*. San Francisco, U.S.A.: Morgan Kaufmann Publishers Inc., 1993. v. 93, n. July, p. 416–423. Cited in the page 31.

GABREL, V.; MURAT, C.; THIELE, A. Recent advances in robust optimization: An overview. *European Journal of Operational Research*, Elsevier, v. 235, n. 3, p. 471–483,2014. Cited in the page 45.

GOUGH, C.; STEINER, I.; SAUNDERS, W. Monitoring. In: *Energy Efficient Servers*.New York, U.S.A.: Springer, 2015. p. 209–268. Cited in the page 86.

HALLAC, D.; LESKOVEC, J.; BOYD, S. Network lasso: Clustering and optimization in large graphs. In: ACM. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Sydney, Australia: ACM Press, 2015. p.
387–396. Cited in the page 22.

HAO, J. et al. Optimization of key parameters of energy management strategy for hybrid electric vehicle using direct algorithm. *Energies*, Multidisciplinary Digital Publishing Institute, v. 9, n. 12, p. 997, 2016. Cited in the page 26.

HARRIS, I.; MUMFORD, C. L.; NAIM, M. M. A hybrid multi-objective approach to capacitated facility location with flexible store allocation for green logistics modeling. *Transportation Research Part E: Logistics and Transportation Review*, Elsevier, v. 66, p.1–22, 2014. Cited in the page 48.

HUANG, H.-Z.; GU, Y.-K.; DU, X. An interactive fuzzy multi-objective optimization method for engineering design. *Engineering Applications of Artificial Intelligence*, Elsevier,v. 19, n. 5, p. 451–460, 2006. Cited 3 times in the pages 23, 27, and 36.

HUBAND, S. et al. A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 10, n. 5, p. 477–506, 2006. Cited in the page 50.

HWANG, C.-L.; LAI, Y.-J.; LIU, T.-Y. A new approach for multiple objective decision making. *Computers & Operations Research*, Elsevier, v. 20, n. 8, p. 889–899, 1993. Cited2 times in the pages 34 and 36.

ICHROME. *Grapheme | Data visualisation*. 2016. Available at: <http://ichrome.com/grapheme>. Accessed on: 19 dez. 2016. Cited in the page 39.

INJA, M. et al. Queued pareto local search for multi-objective optimization. In: SPRINGER. *International Conference on Parallel Problem Solving from Nature*. Ljubljana,Slovenia: Springer, 2014. p. 589–599. Cited in the page 79.

INSELBERG, A. A visual excursion into parallel coordinates. In: *Man-Machine Interactions 3*. Gliwice, Poland: Springer, 2014. p. 43–52. Cited in the page 37.

JMP. *Scatterplot Matrix*. 2016. Available at: <http://www.jmp.com/support/help/Scatterplot_Matrix.shtml>. Accessed on: 20 dez. 2016. Cited in the page 39.

JUANG, C.-F. A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Citeseer, v. 34, n. 2, p. 997–1006, 2004. Cited in the page 22.

KEERATIVUTTITUMRONG, N. et al. Multi-objective co-operative co-evolutionary genetic algorithm. *Parallel Problem Solving from Nature - PPSN VII*, Springer, p. 288–297, 2002. Cited in the page 47.

KELLEY, C. Chapter 38: Implicit filtering and hidden constraints. In: *Advances and Trends in Optimization with Engineering Applications*. Philadelphia, U.S.A.: SIAM, 2017.
p. 507–517. Cited in the page 26.

KELLEY, C. T. *Iterative methods for optimization*. Philadelphia, U.S.A.: SIAM, 1999.Cited in the page 26.

KELNER, V. et al. A hybrid optimization technique coupling an evolutionary and a local search algorithm. *Journal of Computational and Applied Mathematics*, Elsevier, v. 215, n. 2, p. 448–456, 2008. Cited in the page 47.

KLEIN, K.; NEIRA, J. Nelder-mead simplex optimization routine for large-scale problems: A distributed memory implementation. *Computational Economics*, Springer, v. 43, n. 4, p. 447–461, 2014. Cited in the page 26.

KURODA, K. et al. A hybrid multi-objective optimization method considering optimization problems in power distribution systems. *Journal of Modern Power Systemsand Clean Energy*, Springer, v. 3, n. 1, p. 41–50, 2015. Cited in the page 48.

LEE, J. W.; KIM, S. H. Using analytic network process and goal programming for interdependent information system project selection. *Computers & Operations Research*,Elsevier, v. 27, n. 4, p. 367–382, 2000. Cited in the page 34.

LIAO, T. et al. Ant colony optimization for mixed-variable optimization problems. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 18, n. 4, p. 503–518, 2014. Cited in the page 22.

LIEFOOGHE, A. et al. On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics*, Springer, v. 18, n. 2, p. 317–352, 2012. Cited 2 times in the pages76 and 77.

LIN, Q. et al. A novel hybrid multi-objective immune algorithm with adaptive differential evolution. *Computers & Operations Research*, Elsevier, v. 62, p. 95–111, 2015. Cited in the page 48.

LIN, W.; MA, Z.; COOPER, P. Thermal performance evaluation and optimal design of buildings with integrated air-based photovoltaic thermal collectors and phase change materials using the hooke-jeeves pattern search method. 2016. Cited in the page 26.

LOZANO, M.; MOLINA, D.; HERRERA, F. Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, p. 1–3, 2010.Cited in the page 40.

MATHWORKS. *Hypervolume approximation - MATLAB Central*. 2015. Available at: <https://www.mathworks.com/matlabcentral/fileexchange/50517-hypervolume-approximation>. Accessed on: 29 mai. 2018. Cited in the page 84.

MERTINS, A.; MEI, T.; KALLINGER, M. Room impulse response shortening/reshaping with infinity-and p-norm optimization. *IEEE Transactions on Audio, Speech, and Language Processing*, v. 18, n. 2, p. 249, 2010. Cited in the page 33.

MESSAC, A. Physical programming: effective optimization for computational design. *AIAA journal*, v. 34, n. 1, p. 149–158, 1996. Cited in the page 34.

MESSAC, A.; ISMAIL-YAHAYA, A.; MATTSON, C. A. The normalized normal constraint method for generating the pareto frontier. *Structural and Multidisciplinary Optimization*, Springer, v. 25, n. 2, p. 86–98, 2003. Cited in the page 32.

MIETTINEN, K. *Nonlinear Multiobjective Optimization, volume 12 of International Series in Operations Research and Management Science*. Dordrecht, Netherlands: KluwerAcademic Publishers, 1999. Cited in the page 30.

MIETTINEN, K. Survey of methods to visualize alternatives in multiple criteria decision making problems. *OR Spectrum*, Springer, v. 36, n. 1, p. 3–37, 2014. Cited 2 times in the pages 37 and 38.

MONTEIRO, W. R.; REYNOSO-MEZA, G. A multi-criteria based approach for the production distribution in the poultry industry. In: *24th ABCM International Congress of Mechanical Engineering*. Curitiba, Paraná, Brazil: ABCM, 2017. Cited 4 times in the pages 58, 60, 64, and 83.

MONTEIRO, W. R.; REYNOSO-MEZA, G. A hybrid optimization strategy with low resource usage for large scale multi-objective problems. In: 21ST INTERNATIONAL MULTICONFERENCE OF THE INFORMATION SOCIETY. *International Conference on High-Performance Optimization in Industry*. Ljubljana, Slovenia: Jozef Stefan Institute,2018. p. 19–22. Cited in the page 67.

NAIDU, K.; MOKHLIS, H.; BAKAR, A. A. Multiobjective optimization using weighted sum artificial bee colony algorithm for load frequency control. *International Journal of Electrical Power & Energy Systems*, Elsevier, v. 55, p. 657–667, 2014. Cited in the page 31.

ORIGINLAB. *Origin: Data Analysis and Graphing Software*. 2016. Available at: <http://www.originlab.com/index.aspx?go=PRODUCTS/Origin>. Accessed on: 12 dez. 2016. Cited in the page 40.

PAULA, M. S. et al. Modelling and multi-objective optimisation for simulation of cyanobacterial metabolism. 2017. Cited in the page 29.

PLOTLY. *Plotly | Make charts and dashboards online*. 2016. Available at: <https://plot.ly>.Accessed on: 16 dez. 2016. Cited in the page 40.

POTTER, M. A.; JONG, K. A. D. A cooperative coevolutionary approach to function optimization. In: SPRINGER. *International Conference on Parallel Problem Solving from Nature*. Jerusalem, Israel: Springer, 1994. p. 249–257. Cited in the page 45.

QU, B. et al. Novel benchmark functions for continuous multimodal optimization with comparative results. *Swarm and Evolutionary Computation*, Elsevier, v. 26, p. 23–34, 2016. Cited in the page 44.

RAO, S. S. *Engineering optimization: theory and practice*. Hoboken, U.S.A.: John Wiley& Sons, 2009. Cited 2 times in the pages 25 and 26.

REYNOSO-MEZA, G. *Controller Tuning by Means of Evolutionary Multiobjective Optimization: A holistic multiobjective optimization design procedure*. Phd Thesis (PhD Thesis) — Universitat Politècnica de València, 2014. Cited 5 times in the pages 9, 29, 30,
40, and 41.

REYNOSO-MEZA, G. et al. Controller tuning using evolutionary multi-objective optimisation: current trends and applications. *Control Engineering Practice*, Elsevier, v. 28, p. 58–73, 2014. Cited in the page 29.

REYNOSO-MEZA, G. et al. Algoritmos evolutivos y su empleo en el ajuste de controladores del tipo pid: Estado actual y perspectivas. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, Elsevier, v. 10, n. 3, p. 251–268, 2013. Citedin the page 36.

ROY, B. The outranking approach and the foundations of electre methods. In: *Readings in Multiple Criteria Decision Aid*. Lisbon, Portugal: Springer, 1990. p. 155–183. Cited inthe page 34.

SANTANA-QUINTERO, L. V.; MONTANO, A. A.; COELLO, C. A. C. A review of techniques for handling expensive functions in evolutionary multi-objective optimization. In: *Computational Intelligence in Expensive Optimization Problems*. Berlin, Germany: Springer, 2010. p. 29–59. Cited in the page 40.

SIGMAXL. *SigmaXL | Create a Scatter Plot Matrix in Excel using SigmaXL*. 2016. Available at: <http://www.sigmaxl.com/ScatterPlotMatrix.shtml>. Accessed on: 15 dez. 2016. Cited in the page 39.

SINDHYA, K.; DEB, K.; MIETTINEN, K. Improving convergence of evolutionary multi-objective optimization with local search: a concurrent-hybrid algorithm. *Natural Computing*, Springer, v. 10, n. 4, p. 1407–1430, 2011. Cited in the page 48.

SOFTWARE, S. *Sliver Data Visualization Software*. 2016. Available at: <http: //www.sliversoftware.com/index.htm>. Accessed on: 10 dez. 2016. Cited in the page 39.

STADLER, W. *Multicriteria Optimization in Engineering and in the Sciences*. New York, U.S.A.: Springer Science & Business Media, 2013. Cited in the page 21.

TORCZON, V. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, SIAM, v. 7, n. 1, p. 1–25, 1997. Cited in the page 26.

VALIRIS, G.; CHYTAS, P.; GLYKAS, M. Making decisions using the balanced scorecard and the simple multi-attribute rating technique. *Performance Measurement and Metrics*, Emerald Group Publishing Limited, v. 6, n. 3, p. 159–171, 2005. Cited in the page 34.

VELDHUIZEN, D. A. V.; LAMONT, G. B. Evolutionary computation and convergence to a pareto front. In: *Late Breaking Papers at the Genetic Programming 1998 Conference*. Madison, U.S.A.: Omni Press, 1998. p. 221–228. Cited in the page 29.

WARD, M. O.; GRINSTEIN, G.; KEIM, D. *Interactive data visualization: foundations, techniques, and applications*. U.S.A.: AK Peters/CRC Press, 2015. Cited in the page 37.

XDAT. *XDAT - A free parallel coordinates software tool*. 2016. Available at: <http://www.xdat.org/>. Accessed on: 25 dez. 2016. Cited in the page 39.

YANG, Z.; TANG, K.; YAO, X. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, Elsevier, v. 178, n. 15, p. 2985–2999, 2008. Cited 3
times in the pages 45, 46, and 47.

ZHOU, A. et al. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, Elsevier, v. 1, n. 1, p. 32–49, 2011. Cited in the page 41.