MONEDA: Scalable Multi–Objective Optimization with a Neural Network–based Estimation of Distribution Algorithm

Luis Martí lmarti@inf.uc3m.es
Jesús García jgherrer@inf.uc3m.es
Antonio Berlanga aberlan@ia.uc3m.es
José M. Molina molina@ia.uc3m.es
Group of Applied Artificial Intelligence,
Department of Informatics, Universidad Carlos III de Madrid.
Av. de la Universidad Carlos III, 22. Colmenarejo 28270 Madrid, Spain.
http://www.giaa.inf.uc3m.es/

Abstract

The extension of estimation of distribution algorithms (EDAs) to the multi–objective domain has led to multi–objective optimization EDAs (MOEDAs). Most MOEDAs have limited themselves to porting single-objective EDAs to the multi–objective domain. Although MOEDAs have proved to be a valid approach, the last point is an obstacle to the achievement of a significant improvement regarding "standard" multi–objective optimization evolutionary algorithms.

Adapting the model–building algorithm is one way to achieve a substantial advance. Most model–building schemes used so far by EDAs employ off–the–shelf machine learning methods. However, the model–building problem has particular requirements that those methods do not meet and even evade.

The focus of this paper is on the model–building issue and how it has not been properly understood and addressed by most MOEDAs. We delve down into the roots of this matter and hypothesize about its causes. To gain a deeper understanding of the subject we propose a novel algorithm intended to overcome the drawbacks of current MOEDAs.

This new algorithm is the multi–objective neural estimation of distribution algorithm (MONEDA). MONEDA uses a modified growing neural gas network for model–building (MB–GNG). MB–GNG is a custom–made clustering algorithm that meets the above demands. Thanks to its custom-made model–building algorithm, the preservation of elite individuals and its individual replacement scheme, MONEDA is capable of scalably solving continuous multi–objective optimization problems. It performs better than similar algorithms in terms of a set of quality indicators and computational resource requirements.

Keywords

Multi–objective optimization problems, estimation of distribution algorithms, model–building problem, neural networks, growing neural gas.

1 Introduction

Most real–world optimization problems involve optimizing more than one goal. This class of problems is known as multi–objective optimization problems (MOPs). In these

problems the optimizer must find one or more feasible solutions that fit the extreme values (either maximum or minimum) of two or more functions subject to a set of constraints. Therefore, an optimizer's solution is a set of equally good, trade–off solutions. The application of evolutionary computation (EC) (**?**) to MOPs has prompted the creation of what has been called multi–objective optimization evolutionary algorithms (MOEAs) (**??**). Although MOEAs have successfully solved many complex synthetic and real–life problems, the majority of research has focused on low dimensional problems (**?**). MOPs have two main dimensions. One concerns the decision variables, and the other has to do with the number of functions to be optimized. Although the increase in the number of decision variables has a direct impact on the computational cost of evaluating the functions, the addition of more functions is a much tougher issue (**?**).

This type of problems, although counterintuitive and hard to visualize for a human decision maker, are not uncommon in real–life engineering practice (c. f. (**?**)).

When advancing towards higher dimensions of the objective space, the optimization algorithms suffer heavily under the curse of dimensionality (**?**), requiring an exponential increase of the available resources (see **??** and **?**, pp. 414–419).

A viable approach to this issue is to employ cutting–edge evolutionary algorithms that would deal with high–dimensional problems more efficiently. Estimation of distribution algorithms (EDAs) (**???**) are good candidates for such tasks. EDAs have been claimed to constitute a paradigm shift in the evolutionary computation field. Instead of applying evolutionary operators, they create a statistical model of the fittest elements of the population in a process known as model–building. This model is then sampled to produce new elements.

The extension of EDAs to the multi–objective domain has led to multi–objective optimization EDAs (MOEDAs) (**?**). So far, most MOEDAs are extensions of single–objective EDAs to the multi–objective domain. Although MOEDAs have proved to be a valid approach to MOPs, this last point is an obstacle to the achievement of a significant improvement regarding "standard" multi–objective optimization evolutionary algorithms.

Current MOEDAs might have a set of properties stopping them from being a substantial improvement on MOEAs. In particular, we have identified three of such issues: the incorrect treatment of data outliers, the loss of population diversity, and the excess computational effort spent on finding an optimal model of the fittest population elements.

These issues can be traced back to the single–objective predecessor of most MOEDAs and their respective model–building algorithms. Most model–building schemes used so far by EDAs employ off–the–shelf machine learning methods. However, the research community in this area has failed to acknowledge that the model–building problem has particular requirements that ready-made methods do not meet and even contradict. Furthermore, when scaling up the number of objectives this situation is frequently aggravated by the implications of the curse of dimensionality.

The data outliers issue is a good example of how the MOEDA community has overlooked the model–building issue. In common machine–learning practice, outliers are considered as noisy or irrelevant data. However, outliers should be kept in the model–building data set, as they represent newly discovered or candidate search space regions and therefore must be explored. In this case, these instances should be at least as equally represented by the model as the others that are forming groups.

Another drawback of most MOEDAs (and most EDAs, for that matter) is the loss

of population diversity. This has already been pointed out, and some proposals have been laid out for addressing the issue (**???**). This loss of diversity could also be traced back to the nature of the model–building algorithm.

The third issue that must be dealt with is the waste of computational resources on finding an optimal description for the subpopulation being modeled. In the model–building case, optimal model complexity can be sacrificed in the interest of a faster algorithm. This is because, in this context, the only requirement is to have a model that is, if not optimal, complex enough to correctly represent the data. This is particularly true when dealing with high–dimensional MOPs, as, in these cases, there will be large amounts of data to be repeatedly processed in every iteration.

We argue that an understanding of the nature of the model–building problem and the application of suitable algorithms is probably the best way of making substantial progress in this area.

In this paper we examine the model–building issue in depth and introduce a novel MOEDA that is specially devised for correctly dealing with this question. The proposed algorithm is called multi–objective neural estimation of distribution algorithm (MONEDA). MONEDA uses a modified growing neural gas (GNG) network (**?**) for model–building (MB–GNG), which is also introduced here. MB–GNG is a custom–made clustering algorithm that meets the above requirements. Thanks to its custom–made model–building algorithm, the preservation of elite individuals and the individual replacement scheme, MONEDA is capable of scalably solving continuous MOPs and perform better than state–of–the–art algorithms in terms of accuracy and efficiency.

The main contributions of this paper can be summarized as follows:

- we identify and analyze a set of properties of current MOEDAs, and their corresponding model–building algorithms, which are more or less incompatible with the optimization task;
- we put forward MONEDA, a MOEDA that is able to cope with the requirements of the task, and;
- we perform a series of comparative experiments to assess whether our proposal actually tackles this issue properly and how it performs with regard to other MOEDAs and MOEAs.

The remainder of this paper first lays the theoretical groundwork underlying MONEDA. Here we also briefly review the major MOEAs and MOEDAs and point out what needs to be improved to make substantial progress in this field. The model–building GNG is then detailed, followed by a description of the MONEDA algorithm. After that, a series of well–known test problems, the DTLZ3, DTLZ6, DTLZ7, WFG1, WFG2 and WFG6 problems (**??**), are solved with MONEDA and a series of other state–of–the–art algorithms. The performance of each algorithm is assessed using standard community–accepted indicators like the convergence indicator, the Pareto–optimal front coverage indicator, the hypervolume indicator and the unary additive ϵ-indicator. The number of optimization functions is scaled up in each step to assess the behavior of the algorithms when exposed to extreme situations. We conclude with some final remarks, comments and lines for future development.

2 Multi–objective Estimation of Distribution Algorithms

The concept of multi–objective optimization refers to the process of finding one or more feasible solutions of a problem that fit the extreme values (either maximum or mini-

mum) of two or more functions at the same time while being subject to a set of constraints. More formally, a multi–objective optimization problem (MOP) can be defined as:

Definition 1 (Multi–objective Optimization Problem)

$$\left.\begin{array}{ll} \text{minimize} & \boldsymbol{F}(\boldsymbol{x}) = \langle f_1(\boldsymbol{x}), \ldots, f_M(\boldsymbol{x}) \rangle \,, \\ \text{subject to} & c_1(\boldsymbol{x}), \ldots, c_C(\boldsymbol{x}) \leq 0 \,, \\ & d_1(\boldsymbol{x}), \ldots, d_D(\boldsymbol{x}) = 0 \,, \\ & \text{with } \boldsymbol{x} \in \mathcal{D} \,, \end{array}\right\} \tag{1}$$

where $\mathcal{D}$ is known as the decision space*. Functions $f_1(\boldsymbol{x}), \ldots, f_M(\boldsymbol{x})$ are the* objective functions*. The image set, $\mathcal{O}$, product of the projection of $\mathcal{D}$ through $f_1(\boldsymbol{x}), \ldots, f_M(\boldsymbol{x})$ is called* objective space *($\boldsymbol{F} : \mathcal{D} \rightarrow \mathcal{O}$). Finally, $c_1(\boldsymbol{x}), \ldots, c_C(\boldsymbol{x}) \leq 0$ and $d_1(\boldsymbol{x}), \ldots, d_D(\boldsymbol{x}) = 0$ express the restrictions set on the values of $\boldsymbol{x}$.*

Generally speaking, this type of problem does not have a single optimal solution. Instead, an algorithm solving the problem defined in (1) should produce a set containing equally good, trade–off, optimal solutions. The optimality of a set of solutions can be defined based on the so-called *Pareto dominance relation* (**?**):

Definition 2 (Pareto Dominance Relation) *For the optimization problem specified in (1) and having $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{D}$. $\boldsymbol{x}_1$ is said to dominate $\boldsymbol{x}_2$ (expressed as $\boldsymbol{x}_1 \prec \boldsymbol{x}_2$) iff $\forall f_j$, $f_j(\boldsymbol{x}_1) \leq f_j(\boldsymbol{x}_2)$ and $\exists f_i$ such that $f_i(\boldsymbol{x}_1) < f_i(\boldsymbol{x}_2)$.*

The solution of (1) is a subset of $\mathcal{D}$ that contains elements that are not dominated by other elements of $\mathcal{D}$.

Definition 3 (Pareto–optimal Set) *The solution of problem (1) is the set $\mathcal{D}^*$ such that $\mathcal{D}^* \subseteq \mathcal{D}$ and $\forall \boldsymbol{x}_1 \in \mathcal{D}^* \nexists \boldsymbol{x}_2 \in \mathcal{D}$ that $\boldsymbol{x}_2 \prec \boldsymbol{x}_1$.*

$\mathcal{D}^*$ is known as the *Pareto–optimal set* and its image in the objective space is called *Pareto–optimal front*, $\mathcal{O}^*$.

It is often impossible to find the explicit formulation of $\mathcal{D}^*$. Generally, an algorithm solving (1) yields a discrete local Pareto–optimal set, $\mathcal{P}^*$, that approximates $\mathcal{D}^*$. The image of $\mathcal{P}^*$ in objective space, $\mathcal{PF}^*$, is known as the local Pareto–optimal front.

2.1 Evolutionary approaches to multi–objective optimization

A variety of methods have been used to address MOPs (**???**). Of these, evolutionary algorithms (EAs) (**??**) have proven to be a valid and competent approach from the theoretical and practical points of view.

"Evolutionary algorithm" is used as a generic term to indicate a population–based metaheuristic optimization algorithm that uses mechanisms inspired by the biological theory of evolution (**?**). Each individual in the population represents a candidate solution for the problem being solved. A fitness assignment function determines how fit or adapted an individual is to its environment, or, in other words, how well the solution represented by the individual performs compared with the rest of the population. Individuals are recombined and improved using evolutionary operators inspired by the natural processes of reproduction, cross–over and mutation.

Single objective EAs have been successfully extrapolated to the multi–objective domain. This has led to what has been called multi–objective optimization evolutionary algorithms (MOEAs) (**??**). Their success is due to the fact that EAs do not make any assumptions about the underlying fitness landscape. Therefore, they are believed to perform consistently well across a wide range of problems, although it has been shown

that they share theoretical limits imposed by the no–free–lunch theorem (**?**). The parallelism of the search process results in another important benefit, because these algorithms can produce a set of equally optimal solutions instead of just one, as many other algorithms do.

The first MOEAs proposed were non–elitist. This group includes the vector evaluated genetic algorithm (VEGA) (**?**), the multi–objective genetic algorithm (MOGA) (**??**), the non–dominated sorting genetic algorithm (NSGA) (**?**) and the niched–Pareto genetic algorithm (NPGA) (**?**) among others. The last three algorithms apply a non–dominated classification of the population, using another alternative type of selection operators.

Elitist approaches to MOEA are more recent techniques. They include notably the elitist NSGA (NSGA–II) (**?**), the strength Pareto evolutionary algorithm (SPEA) (**?**), the improved SPEA (SPEA2) (**?**), Pareto–archived evolution strategy (PAES) (**??**) and the Pareto envelope–based selection algorithm (PESA) (**??**), among many others.

2.2 Estimation of distribution algorithms

Estimation of distribution algorithms (EDAs) have been claimed to be a paradigm shift in the field of evolutionary computation. Like EAs, EDAs are population-based optimization algorithms. In EDAs, however, instead of applying the evolutionary operators to the population, a statistical model of the most promising subset of the population is built. This model is then sampled to produce new individuals that are merged with the original population according to a given substitution policy. Because of this model–building feature, EDAs have also been called probabilistic–model–building genetic algorithms (PMBGAs) (**?**). A framework similar to EDAs is proposed by the iterated density estimation evolutionary algorithms (IDEAs) (**?**).

The introduction of machine learning techniques implies that these new algorithms do not have the biological plausibility of their predecessors. In return, they gain the capacity of scalably solving many challenging problems, often significantly outperforming standard EAs and other optimization techniques. The early EDAs were intended for combinatorial optimization, but they have since been extended to continuous domains (see **?** for a review).

2.3 Multi–objective estimation of distribution algorithms

Multi–objective optimization EDAs (MOEDAs) (**?**) are the extensions of EDAs to the multi–objective domain. Most MOEDAs are a modification of existing EDAs whose fitness assignment strategy has been replaced by a previously existing method used by MOEAs.

Before going on to discuss the proposal presented in this paper, let us analyze the current approaches to this topic and briefly highlight their strengths and weaknesses. Note that this is not meant to be a comprehensive account of EDAs but we will instead focus on EDAs and their extrapolation to the multi–objective domain. Therefore, we will enumerate only those algorithms of interest. Table 1 summarizes the properties of the MOEDAs described here.

2.3.1 Algorithms based on Bayesian networks

A very popular foundation for MOEDAs is a range of EDAs that builds the population model using a Bayesian network. The Bayesian optimization algorithm (BOA) (**?**), the estimation of Bayesian network algorithm (EBNA) (**?**) and the learning factorized distribution algorithm (LFDA) (**?**) are members of this group.

The exhaustive synthesis of a Bayesian network (**?**) from the algorithm's population is a NP–hard problem. Therefore, the intention behind the above approaches is to provide heuristics for building the network of reasonable computational complexity. BOA uses a so–called K2 metric, based on the Bayesian Dirichlet metric (**?**), to assess the network quality. A simple greedy algorithm that adds edges in each iteration. EBNA, on the other hand, has been tested with different metrics, like the Bayesian information criterion (BIC) (**?**), K2+penalty and on testing conditional (in)dependencies between variables. LFDA relies on the greedy algorithm introduced by FDA. The complexity of the learnt model is controlled by the BIC criterion in conjunction with the maximum number of incoming edges in the network constraint.

Most BOA-based MOEDAs are a combination of a BOA-based model–building scheme with an already existing Pareto–based fitness assignment. This is the case of the multi–objective BOA (mBOA) (**?**) that exploits the fitness assignment used in NSGA–II. A later algorithm based on hierarchical BOA (hBOA) (**???**), called mhBOA (**?**) also used the same form of fitness assignment. A similar idea is proposed in (**??**), where the mixed BOA (mBOA) (**?**) is combined with the SPEA2 selection scheme to form the multi–objective mBOA (mmBOA).

Multi–objective real BOA (MrBOA) (**?**) also extends a preexisting EDA, in this case the real BOA (rBOA) (**?**). RBOA performs a proper problem decomposition by means of a Bayesian factorization and probabilistic building–block crossover. To do this, it employs mixture models at the level of subproblems. MrBOA combines the fitness assignment of NSGA–II with rBOA.

2.3.2 Multi–objective Mixture–Based Iterated Density Estimation Evolutionary Algorithm (MIDEA)

Another approach to modeling the subset with the best population elements is to apply a distribution mixture approach. In a series of papers, Bosman and Thierens (**??????**) proposed several variants of their multi–objective mixture–based iterated density estimation algorithm (MIDEA). They are based on their IDEA framework. Bosman and Thierens proposed a novel Pareto–based and diversity–preserving fitness assignment function. The model construction is inherited from the single–objective version. The proposed MIDEAs considered several types of probabilistic models for both discrete and continuous problems. A mixture of univariate distributions and a mixture of tree distributions were used for discrete variables. A mixture of univariate Gaussian models and a mixture of multivariate Gaussian factorizations were applied for continuous variables. An adaptive clustering method was used to determine the capacity required to model a population.

MIDEAs do not place any constraints on the location of the centers of the distributions. Consequently, the MIDEA clustering mechanism does not provide a specific mechanism to ensure equal coverage of the Pareto–optimal front if the number of representatives in some parts of the front is much larger than the number of representatives in some other parts.

The clustering algorithms applied to do this include the randomized leader algorithm (**?**), the k–means algorithm (**?**) and the expectation maximization algorithm (**?**).

2.3.3 Regularity Model–Based Multi–Objective Estimation of Distribution Algorithm (RM–MEDA)

The regularity model–based multi–objective estimation of distribution algorithm (RM–MEDA) (**??**) is based on the regularity property derived from the Karush–Kuhn–Tucker condition. This means that, subject to certain constraints, it can be induced

Table 1: Summary of the main characteristics of the different MOEDAs discussed and our proposal, MONEDA.

MOEDA	Domain	Fitness	Model–Building	Original EDA
mBOA	combinatorial	NSGA–II	Bayesian	BOA
mhBOA	combinatorial	NSGA–II	Bayesian	hier. BOA
mmBOA	comb. + cont.	SPEA2	Bayesian	mixed BOA
MrBOA	continuous	NSGA–II	Bayesian	real BOA
Naïve MIDEA	continuous	proprietary	Univ. dists	IDEA
RM–MEDA	continuous	NSGA–II	Regularity	—
MOPED	continuous	NSGA–II	Parzen	—
MONEDA	continuous	NSGA–II	MB–GNG	—

that the Pareto–optimal set, $\mathcal{D}^*$, of a continuous multi–objective optimization problem is a piecewise continuous $(M-1)$–dimensional manifold, where M is the number of objectives (**??**).

At each iteration, RM–MEDA models the promising area of the decision space by a probability distribution, whose centroid is a $(M-1)$–dimensional piecewise continuous manifold. The local principal component analysis algorithm (**?**) is used to build such a model. New trial solutions are sampled from the model built thus. Again, the fitness assignment used is the one proposed by NSGA–II. The main drawback of this algorithm is its high computational complexity.

2.3.4 Multi–Objective Parzen EDA (MOPED)

Multi–objective Parzen EDA (MOPED) (**??**) uses the NSGA–II ranking method and the Parzen estimator (**?**) to approximate the probability density of solutions lying on the Pareto front. The proposed algorithm has been applied to different types of test case problems, and results show a good performance of the overall optimization procedure in terms of the number of function evaluations.

3 Scalability and the for of a new MOEDA

One topic that has yet to be properly dealt with in the MOEA/MOEDA field is algorithm scalability (**??**) or what has been termed *many–objective problems* (**??**). The scalability issue for these algorithms is stated in terms of two magnitudes:

1. the decision space dimension, $\mathcal{D}$, that is, the number of variables that are involved in the problem, and;

2. the objective space dimension, $\mathcal{O}$, or in other words, the number of objective functions to be optimized.

A critical quantity is the objective space dimension as it has been shown experimentally to have an exponential relation to the optimal size of the population (see **???** and **?**pp. 414–419>Deb01e). This implies that an exponential amount of resources need to be made available to an optimization algorithm, as the of the number of objective functions increases.

As already mentioned, a number of works (**????**) have targeted the minimization of the number of objective functions to make the problem less complex. Although this

research provide a most useful tool for relieving the workload of a given problem, it does not ultimately address the key issue: how to create MOEDAs capable of efficiently solving high–dimensional problems.

This question can be reduced to the problem of how to handle a relatively small population that does not adequately represent non–dominated and dominated individuals. This renders approaches based on Pareto dominance useless. Therefore, in order to achieve a sizable improvement in the scalability of these algorithms, they need to be armed with an efficient, scalable and robust fitness assignment function and with a purpose–built model–building algorithm that promotes search and drives the population towards newly found sub–optimal regions in the objective space.

3.1 Improving the fitness assignment

Finding a better fitness assignment function is a complex issue. As previously discussed, there is an exponential explosion in the amount of resources required as dimensions grow. This growth means that the mutual comparison and/or sorting processes that are part of fitness assignment become very time consuming. One possible solution is to bypass the exponential relation, but this will probably lead to a situation where many individuals are non-dominated, and therefore it is impossible to find a correct direction for the search.

A number of works (**??????**) have proposed the use of performance indicators, in particular the hypervolume indicator (**??**), as fitness assignment functions. This is a promising line of research, as these approaches offer a solution in the situations where most of the population is non–dominated and it is not possible to find a search direction. However, this indicator has been shown to be very computationally complex to compute (see section 5.2.2 for further details). Still, some recent developments in this direction (**???**) have opened a door to their application in high–dimensional problems.

3.2 Addressing the model–building issue

Improving the model–building algorithm looks to be a promising direction for research, as it has, to the best of our knowledge, not been properly addressed. So far, MOEDA approaches have mostly used off–the–shelf machine learning methods. The characteristics of the task at hand are different to the features of the tasks for which those methods were originally designed.

These characteristics, as the introduction of the paper suggests, are:

- incorrect treatment of data outliers;
- loss of population diversity, and
- excess of computational effort for finding an optimal model of the fittest elements of the population.

These undesirable properties were determined as a result of a set of preliminary studies (**??**), where we compared the behavior of a set of model–building algorithms under the same MOEDA framework. That study found that, in high–dimensional problems and under the same experimental conditions, statistically robust algorithms, like those commonly used for the synthesis of Bayesian networks (**?**), were outperformed by "less robust" approaches like k–means algorithm (**?**) or the randomized leader algorithm (**?**).

The cause of this behavior can be attributed to the fact that statistically rigorous methods are not specifically meant for the problem we are dealing with here. These behaviors, although justified in the original field of application of the algorithms, could

be an obstacle to process performance in terms of both accuracy and resource consumption. Two of the above behaviors are important in this respect: incorrect treatment of outliers and overexpenditure of resources on finding the optimal model structure or topology.

In the statistical and machine learning areas, data instances that are relatively isolated or different to the greater masses of data are known as outliers. Historically, these outliers are treated as unrepresentative, noisy or bogus data.

In model–building, however, all the available data are known beforehand to be valid, as they represent the best part of the current population. Therefore, no points must be disregarded. Instead, these outliers are essential, as they represent unexplored or recently discovered areas of the current Pareto–optimal front. They should not only be preserved but, perhaps, even reinforced.

A model–building algorithm that primes outliers might actually accelerate the search process and alleviate the exponential dimension–population size dependency ratio.

Some of the results on diversity loss in other MOEDAs back up this reasoning (**???**). This loss of diversity can be traced back to the above issue of outliers in model–building algorithms. The repetitive application of an algorithm that disregards outliers tends to generate more individuals in more densely represented areas of the search space. Although there have been some proposals to circumvent this problem, we take the view that the ultimate solution is the use of an adequate algorithm.

The root cause of most standard methods disregarding outliers can be traced back to the error–based learning used in those methods. Error-based learning minimizes a dataset–wise error. For this reason, infrequent or poorly represented elements are sacrificed in order to achieve a better overall error.

Similarly, there is no need for the model–building algorithm to find the least complex accurate data model. Even so, most of the current approaches dedicate a sizable effort to finding the optimal model complexity using minimum description length (**?**), structural risk minimization (**?**), Bayesian information criterion (**?**) or other similar heuristics.

For a thorough discussion of these matters, see (**?**), .

By way of a conclusion to this analysis, and based on the recommendations and groundwork of our earlier work, there is a clear need for a novel MOEDA that properly addresses the model–building issue. In the next section we introduce a MOEDA that uses a purpose–built model–building algorithm to overcome the problems described here.

4 Multi–objective Neural EDA

The multi–objective optimization neural estimation of distribution algorithm (MONEDA) combines the NSGA–II fitness assignment and a model builder that uses a modification of the growing neural gas network designed for model–building (MB–GNG). The MB–GNG network is a custom–made model–building algorithm devised to cope with the task specifications.

The NSGA–II fitness assignment is very well understood and has relatively low computational cost, which explains why it was selected. Also, a more direct comparison with similar MOEDA approaches using this assignment strategy, since it is the most popular choice. Let us point out, however, that recent advances in indicator–based fitness assignment have, as already mentioned, rendered that approach an attractive choice. We have started to explore this direction (**?**).

Briefly, MONEDA was devised with the following properties in mind:

- *scalability*: MONEDA is expected to outperform similar algorithms when solving many–objective problems;
- *elitism*: as its has proven itself to be a very advantageous feature in evolutionary algorithms, and;
- *diversity preservation*: in spite of promoting the preservation of the fittest solutions, it is also essential that the population remains as diverse as possible.

4.1 Model–building with a modified growing neural gas network

Clustering algorithms (**????**) have been used as part of EDA and MOEDA model–building algorithms. However, as discussed in the previous section, a purpose-built algorithm might be a way of achieving a significant improvement in this field.

After surveying the literature for suitable candidates, we chose the growing neural gas (GNG) network (**?**) as a starting point . GNG networks are intrinsic self–organizing neural networks based on the neural gas (**?**) model. This model relies on a competitive Hebbian learning rule (**?**).

Of the vast number of existing clustering methods, we decided to base our approach on GNG because of its interesting properties, in particular:

- the network is sensitive to outliers (**?**), which, although undesirable in standard applications, is a positive feature for model building;
- the network grows to adapt itself automatically to the complexity of the problem being solved;
- it yields fast convergence to low distortion errors, and these errors are better than those yielded by "standard" algorithms like k-means clustering, maximum–entropy clustering and Kohonen's self–organizing feature maps (**?**);
- its learning rule is based on a stochastic gradient descent on an explicit energy surface (**?**);
- although it benefits from the topological ordering of the nodes, it does not suffer the problem associated with Kohonen networks, where a node can pull its neighbors into invalid or non–representative locations of the input space, and;
- the addition of a cluster repulsion mechanism fosters the exploration of the input space, assuring that each cluster represents a distinct region of the space.

A GNG network creates an ordered topology of input classes and associates a cumulative error to each. The topology and the cumulative errors are conjointly used to determine how new classes should be inserted. Using these heuristics the model can fit the network dimension to the complexity of the problem being solved. GNG was originally meant to solve unsupervised learning problems (i.e. clustering and vector quantization); it was extended to supervised RBF networks (**??**) for the incremental generation of neuro–fuzzy systems (**?**).

Our model–building GNG (MB–GNG) is an extension of the original GNG. It introduces a cluster repulsion term that fosters a better spread of the clusters across the training dataset, as explained by **?**.

MB–GNG is a one–layer network that defines each class as a local Gaussian density and adapts them using a local learning rule. The layer contains a set of nodes $\mathcal{C} = \{c_1, \ldots, c_{N^*}\}$, with $N_0 \leq N^* \leq N_{\max}$. Here N_0 and $N_{\max}$ represent an initial and maximal number of nodes in the network.

A node c_i describes a local multivariate Gaussian density that consists of a center, $\boldsymbol{\mu}_i$, standard deviations vector, $\boldsymbol{\sigma}_i$. It also has an accumulated error, ξ_i, and a set of edges that define the set of topological neighbors of c_i, $\mathcal{V}_i$. Each edge has an associated age, ν_{ij}.

MB–GNG creates a quantization of the inputs space using a modified version of the GNG algorithm and then computes the deviations associated with each node.

The dynamics of a GNG network consists of three concurrent processes: network adaptation, node insertion and node deletion. The combined use of these three processes renders GNG training Hebbian in spirit (**?**).

The network is initialized with N_0 nodes with their centers set to randomly chosen inputs. A training iteration starts after an input $\boldsymbol{x}$ is randomly selected from the training data set. Then two nodes are selected for being the closest ones to $\boldsymbol{x}$. The *best matching node*, c_b,

$$b = \underset{i=1,..,N^*}{\arg\min}\, d\left(\boldsymbol{\mu}_i, \boldsymbol{x}\right), \tag{2}$$

is the closest node to $\boldsymbol{x}$. Consequently, the *second best matching node*, $c_{b'}$, is determined as

$$b' = \underset{i=1,..,N^*; i \neq b}{\arg\min}\, d\left(\boldsymbol{\mu}_i, \boldsymbol{x}\right). \tag{3}$$

Here $d\left(\boldsymbol{a}, \boldsymbol{b}\right)$ is a distance metric. For this research we used $d(\cdot)$ defined as

$$d\left(\boldsymbol{a}, \boldsymbol{b}\right) = \left\|\boldsymbol{a} - \boldsymbol{b}\right\|. \tag{4}$$

If $c_{b'}$ is not a neighbor of c_b then they are linked by a new edge $\mathcal{V}_b = \mathcal{V}_b \cup \{c_{b'}\}$ with zero age, $\nu_{bb'} = 0$. If, on the other hand, $c_{b'} \in \mathcal{V}_b$, the age of the corresponding edge is reset $\nu_{bb'} = 0$.

At this point, the age of all edges is incremented by one. If an edge is older than the maximum age, $\nu_{ij} > \nu_{\max}$, then the edge is removed. If a node becomes isolated from the rest, it is also deleted.

Clustering error is then added to the best matching node error accumulator,

$$\Delta\xi_b = d\left(\boldsymbol{\mu}_i, \boldsymbol{x}\right)^2. \tag{5}$$

After that, learning takes place in the best matching node and its neighbors with rates ϵ_{best} and ϵ_{vic} ($\epsilon_{\text{best}} > \epsilon_{\text{vic}}$), respectively. These two rates gate the movement of the centers of the nodes involved towards the current input $\boldsymbol{x}$. Figure 1 shows a diagram of this process.

For c_b, adaptation follows the rule originally used by GNG,

$$\Delta\boldsymbol{\mu}_b = \epsilon_{\text{best}}\left(\boldsymbol{x} - \boldsymbol{\mu}_b\right). \tag{6}$$

However, for the neighbors of c_b, a cluster repulsion term (**?**) is added to the original formulation. For such nodes the learning rule combines a movement towards the input and a repulsion term that takes into account the distance to neighboring nodes. This repulsion term avoids the meaningless concentration of nodes in the data space and, therefore, promotes a proper representation of the data set with fewer nodes.

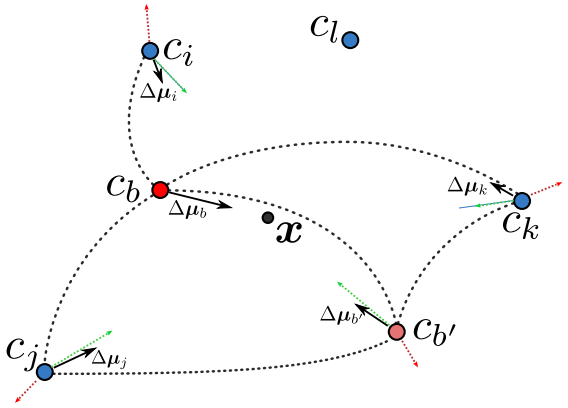


Figure 1: Schematic representation of MB–GNG learning. Node neighborhood edges are represented by dotted arcs. The center of the best matching node, $\boldsymbol{\mu}_b$, is modified according to (6). For the neighbors of c_b, learning takes place according to 7. For those nodes the learning rule combines a movement towards the input (represented in green) and a repulsion term which takes into account the distance to neighboring nodes (in red). This repulsion term avoids the concentration of nodes in the same region of the data space. No learning takes place on nodes disconnected from c_b, like c_l

Following that, the learning rule for those nodes can be expressed as, $\forall c_v \in \mathcal{V}_b$,

$$\Delta\boldsymbol{\mu}_v = \epsilon_{\text{vic}}\left(\boldsymbol{x} - \boldsymbol{\mu}_v\right) + \beta e^{\left(-\frac{d(\boldsymbol{\mu}_v,\boldsymbol{\mu}_b)}{\zeta}\right)} \frac{\sum_{c_u \in \mathcal{V}_b} d\left(\boldsymbol{\mu}_u, \boldsymbol{\mu}_b\right)}{|\mathcal{V}_b|} \frac{\left(\boldsymbol{\mu}_v - \boldsymbol{\mu}_b\right)}{d\left(\boldsymbol{\mu}_v, \boldsymbol{\mu}_b\right)}. \tag{7}$$

This approach was already used as part of the robust GNG (**?**), and has proved to be useful for obtaining a good spread of the clusters in the input space. **?** stated that the adaptation rule is not sensitive to its parameters. Here β is an integral multiplier that defines the amplitude of the repulsive force, whereas ζ controls the weakening rate of the repulsive force regarding the distance between node centers. We have set the centers to $\beta = 2$ and $\zeta = 0.1$ as suggested by **?**.

After a given number, T_+, of dataset iterations (epochs in neural networks terminology) have taken place, it can be presumed that the error accumulators, ξ_i have stored enough information. This information is used to determine where to add new nodes to the network. In particular, if the current iteration is an integer multiple of T_+ and the network has not reached its maximum size ($N^* < N_{\max}$), then a new node is inserted in the network.

First, the node with the largest error, c_e, is selected. Then, the worst node in its neighborhood, $c_{e'}$, is located. Then N^* is incremented, and the new node, c_{N^*}, is inserted between the two nodes,

$$\boldsymbol{\mu}_{N^*} = 0.5\left(\boldsymbol{\mu}_e + \boldsymbol{\mu}_{e'}\right). \tag{8}$$

The edge between c_e and $c_{e'}$ is removed, and two new edges are created to connect c_{N^*} with c_e and $c_{e'}$. The accumulated errors are decreased

$$\xi_e = \delta_I \xi_e,\ \xi_{e'} = \delta_{\text{I}} \xi_{e'}, \tag{9}$$

Parameters:
- N_0 and $N_{\max}$, bounds on the number of nodes.
- $\nu_{\max}$, maximum edge age.
- ϵ_{best} and ϵ_{vic}, learning rates.
- T_+, number of dataset iterations before node insertion process.
- δ_{I}, δ_{G}, error redistribution rates.
- ρ, stopping threshold.

Initialize $t \leftarrow 0$.
Randomly select N_0 elements from the dataset and initialize the same number of nodes using those elements as centers $\boldsymbol{\mu}$, and empty $\mathcal{V}$s.
repeat
  Determine best matching node, c_b, following (2).
  Determine second best matching node, $c_{b'}$, following (3).
  if $c_{b'} \notin \mathcal{V}_b$ **then**
    Make c_b and $c_{b'}$ neighbors,

$$\mathcal{V}_b = \mathcal{V}_b \cup \{c_{b'}\}; \mathcal{V}_{b'} = \mathcal{V}_{b'} \cup \{c_b\}.$$

  end if
  Neighborhood edge age is set to $\nu_{bb'} = 0$.
  Update c_b error accumulator, ξ_b, according to (5).
  Learning takes place in c_b as specified in (6).
  $\forall c_{\text{vic}} \in \mathcal{V}_b$ learning is carried out according to (7).
  if $t \bmod T_+ = 0$ **and** $N^* < N_{\max}$ **then**
    Determine node that has the largest accumulated error, c_e, and is the worst in its neighborhood, $c_{e'}$, $c_{e'} \in \mathcal{V}_e$.
    Dissolve edge between c_e and $c_{e'}$,

$$\mathcal{V}_e = \mathcal{V}_e \setminus \{c_{e'}\}; \mathcal{V}_{e'} = \mathcal{V}_{e'} \setminus \{c_e\}.$$

    Create a new node between c_e and $c_{e'}$, as in (8) and (10).
    Decrease c_e and $c_{e'}$ accumulated errors, as expressed in (9).
    Decrease the errors of the remaining nodes, following (11).
  end if
until inequality (12) holds.
Compute the unbiased estimator of the deviations.

Figure 2: The algorithm of MB–GNG.

by a rate $0 \leq \delta_{\text{I}} \leq 1$. The error of the newly created node is computed as

$$\xi_{N^*} = 0.5(\xi_e + \xi_{e'}). \tag{10}$$

Finally, the errors of all nodes are decreased by a factor δ_{G},

$$\xi_i = \delta_{\text{G}} \xi_i, \; i = 1, .., N^*. \tag{11}$$

When to stop GNG learning is a non–trivial issue shared by other clustering algorithms and all reiterative heuristic algorithms. As the main priority here is to cover as

much of the input space as possible, we will stop if, at the end of a learning epoch, the standard deviation of the accumulated errors is less than a set threshold, ρ,

$$\sqrt{\frac{1}{N^*}\sum_{i=1}^{N^*}(\xi_i - \overline{\xi})^2} < \rho. \tag{12}$$

This means that it will stop when the outliers are as well represented as possible.

After training has ended, the deviations, $\boldsymbol{\sigma}_i$, of the nodes must be computed. To do this we employ the unbiased normal estimator of the deviations (**?**) detailed in the following algorithm:

Set $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{N^*} = \mathbf{0}$ and $n_1, \ldots, n_{N^*} = 0$.
for all $\boldsymbol{x} \in \Psi$ **do**
  Determine the closest node, c_c to $\boldsymbol{x}$.
  $\boldsymbol{s}_c = \boldsymbol{s}_c + (\boldsymbol{x} - \boldsymbol{\mu}_c)^2$.
  $n_c = n_c + 1$.
end for
Compute the deviations as $\boldsymbol{\delta}_i = \sqrt{\frac{\boldsymbol{s}_i}{n_i}}$.

The local Gaussian densities resulting from the above algorithm can be combined to synthesize the Gaussian mixture with parameters $\boldsymbol{\Theta}$,

$$P(\boldsymbol{x}|\boldsymbol{\Theta}) = \frac{1}{N^*}\sum_{i=1}^{N^*} P(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i)\,. \tag{13}$$

Each Gaussian density is formulated as

$$P(\boldsymbol{x}|\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}_i|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu_i})^\top \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i)\right), \tag{14}$$

with the covariance matrices $\boldsymbol{\Sigma}_i$ defined as a diagonal matrix with its non–zero elements set to the values of the deviations $\boldsymbol{\sigma}_i$,

$$\boldsymbol{\Sigma}_i = \boldsymbol{I}\boldsymbol{\sigma}_i\,. \tag{15}$$

Here $|\Sigma_i|$ is the determinant of $\boldsymbol{\Sigma}_i$, $\boldsymbol{I}$ is the identity matrix and, again, n is the dimension of $\boldsymbol{x}$.

The Gaussian mixture can be used by the EDA to generate new individuals. These new individuals are created by sampling the $P(\boldsymbol{x}|\boldsymbol{\Theta})$. Many researchers have dealt with the generation of randomly distributed numbers that follow a given distribution. It has been thoroughly described, for example, by **?**. In our case, we applied the Box–Muller transformation (**?**). This transformation converts uniformly distributed random variables to a new set of random variables with a Gaussian distribution.

See Figure 2 for a summary of the MB–GNG algorithm.

4.2 The MONEDA algorithm

MONEDA maintains a population, $\mathcal{P}_t$, of n_{pop} individuals, where t is a given iteration. The algorithm's workflow is similar to other EDAs (see Figure 3). It starts with a random initial population $\mathcal{P}_0$ of individuals. It then proceeds to sort the individuals using the NSGA–II fitness assignment function.

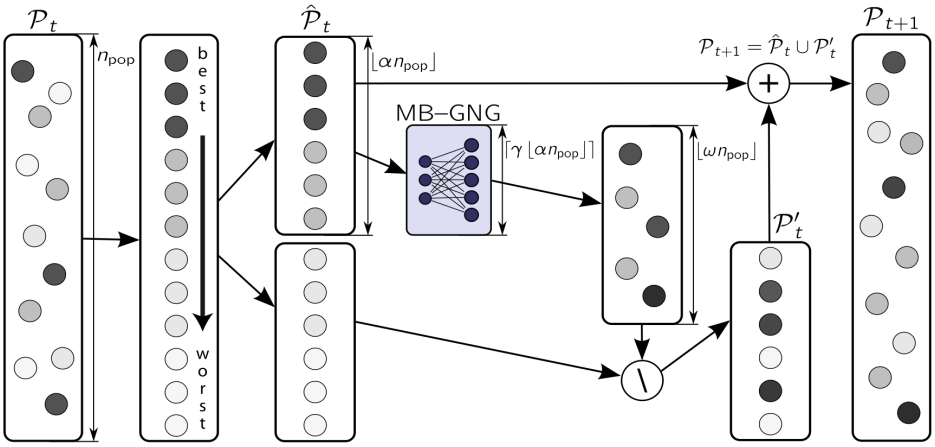


Figure 3: Diagram of the MONEDA algorithm. At iteration t the population is ranked using the fitness assignment function. Then a population subset $\hat{\mathcal{P}}_t$ containing the best $\lfloor \alpha n_{\text{pop}} \rfloor$ elements of $\mathcal{P}_t$ is extracted. A MB–GNG network is trained with the elements of $\hat{\mathcal{P}}_t$. $\lfloor \omega n_{\text{pop}} \rfloor$ new individuals are sampled from the neural network. These individuals substitute the same number of randomly selected elements of $\mathcal{P}_t \setminus \hat{\mathcal{P}}_t$. The resulting set is then combined with $\hat{\mathcal{P}}_t$ to form the population of the next iteration, $\mathcal{P}_{t+1}$.

NSGA–II fitness assignment is a two-phase process. First, individuals are ranked according to the dominance relations by which they are linked. Then, individuals with the same domination rank are then compared using a local crowding distance.

The first step consists of classifying the individuals in a series of categories, $\mathcal{F}_1, \ldots, \mathcal{F}_L$. Each of these categories stores individuals that are only dominated by the elements of the previous categories,

$$\begin{aligned} \forall \boldsymbol{x} \in \mathcal{F}_i : \quad & \exists \boldsymbol{y} \in \mathcal{F}_{i-1} \text{ such that } \boldsymbol{y} \prec \boldsymbol{x}, \text{ and;} \\ & \nexists \boldsymbol{z} \in \mathcal{P}_t \setminus (\mathcal{F}_1 \cup \ldots \cup \mathcal{F}_{i-1}) \text{ that } \boldsymbol{z} \prec \boldsymbol{x}; \end{aligned} \tag{16}$$

with $\mathcal{F}_1$ equal to $\mathcal{P}_t^*$, the set of non–dominated individuals of $\mathcal{P}_t$.

After all individuals have been ranked, they are assigned a local crowding distance. The use of this distance primes individuals that are more isolated from others. The assignment process goes as follows,

for all category sets $\mathcal{F}_l$, having $f_l = |\mathcal{F}_l|$ **do**
  for all individuals $\boldsymbol{x}_i \in \mathcal{F}_l$ **do**
    $d_i = 0$.
  end for
  for all objective functions $m = 1, \ldots, M$ **do**
    $\boldsymbol{I} = \text{sort}\,(\mathcal{F}_l, m)$ (generate index vector in ascending order).
    $d_{I_1}^{(l)} = d_{I_{f_l}}^{(l)} = \infty$.
    for $i = 2, \ldots, f_l - 1$ **do**
      Update the remaining distances as

$$d_i = d_i + \frac{f_m\left(\boldsymbol{x}_{I_{i+1}}\right) - f_m\left(\boldsymbol{x}_{I_{i-1}}\right)}{f_m\left(\boldsymbol{x}_{I_{f_l}}\right) - f_m\left(\boldsymbol{x}_{I_1}\right)}.$$

end for
end for
end for

Here the $\mathrm{sort}\,(\mathcal{F}, m)$ function produces a vector $\boldsymbol{I}$ of indexes ranked in ascending order with respect to the value of objective function f_m.

They are sorted by their individual ranks and local distances, using the following operator:

Definition 4 (Crowded Comparison Operator) *An individual $\boldsymbol{x}_i$ is better than $\boldsymbol{x}_j$ if:*

- *$\boldsymbol{x}_i$ has a better rank: $\boldsymbol{x}_i \in \mathcal{F}_k$, $\boldsymbol{x}_j \in \mathcal{F}_l$ and $k < l$, or;*
- *if $k = l$ and $d_i > d_j$.*

A set $\hat{\mathcal{P}}_t$ containing the best $\lfloor \alpha \left| \mathcal{P}_t \right| \rfloor$ elements is extracted from the sorted version of $\mathcal{P}_t$,

$$\left| \hat{\mathcal{P}}_t \right| = \lfloor \alpha \left| \mathcal{P}_t \right| \rfloor = \lfloor \alpha n_{\mathrm{pop}} \rfloor \ . \tag{17}$$

A MB–GNG network is then trained using $\hat{\mathcal{P}}_t$ as its training data set. To get a controlled relation between the size of $\hat{\mathcal{P}}_t$ and the maximum size of the network, $N_{\max}$, these two sizes are bounded by the rate $\gamma \in (0, 1]$,

$$N_{\max} = \left\lceil \gamma \left| \hat{\mathcal{P}}_t \right| \right\rceil = \lceil \gamma \lfloor \alpha n_{\mathrm{pop}} \rfloor \rceil \ . \tag{18}$$

The trained GNG network is a model of $\hat{\mathcal{P}}_t$. The network can be interpreted as a Gaussian mixture, as explained in the previous section. Therefore, it can be used to sample new individuals. In particular, $\lfloor \omega \left| \mathcal{P}_t \right| \rfloor$ new individuals are synthesized.

Each individual substitutes another randomly selected individual from the section of the population not used for model–building $\mathcal{P}_t \setminus \hat{\mathcal{P}}_t$. The resulting set is then united with the best elements, $\hat{\mathcal{P}}_t$, to form the population of the next iteration $\mathcal{P}_{t+1}$. Some other substitution strategies could be used in this step. For example, the new individuals could substitute the worst individuals of $\mathcal{P}_t \setminus \hat{\mathcal{P}}_t$. We opted for the above approach because it promotes diversity and avoids stagnation.

Iterations are repeated until a given stopping criterion is met. The output of the algorithm is a subset of $\mathcal{P}_t$ that contains the non–dominated solutions, $\mathcal{P}_t^*$.

Figure 4 gives an algorithmic outline of MONEDA.

5 Assessing MONEDA

A key part of this work is to understand how MONEDA performs in practical situations, and its outcome compared with similar state–of–the–art algorithms. MONEDA embeds the hypothesis related to the model–building issue presented in the above theoretical discussion. Therefore, an analysis of the experimental results is indispensable for gaining a better understanding of the issue. For this reason, we now focus on solving a set of well–known problems with a selected set of the previously discussed evolutionary multi–objective optimizers, namely, naïve MIDEA (**?**), MrBOA (**?**), RM–MEDA (**?**), MOPED (**?**), NSGA–II (**?**), SPEA2 (**?**) and, of course, MONEDA.

We will now describe the experimental setup of our study in detail. First, we discuss the test problems used and the performance indicators applied. We then depict the hardware and software configurations and the choice of the initial parameters of the applied algorithms.

MB–GNG parameters: N_0, $\nu_{\max}$, ϵ_{best}, ϵ_{vic}, T_+ δ_{I}, δ_{G} and ρ.
MONEDA parameters: n_{pop}, α, γ and ω.
$t \leftarrow 0$.
Randomly generate the initial population $\mathcal{P}_0$ with n_{pop} individuals.
repeat
  Sort $\mathcal{P}_t$ individuals with regard to the crowded comparison operator.
  Extract first $\alpha\,|\mathcal{P}_t|$ elements the sorted $\mathcal{P}_t$ to $\hat{\mathcal{P}}_t$.
  Train MB–GNG network with $\hat{\mathcal{P}}_t$ training data set and $N_{\max} = \left\lfloor \gamma \left| \hat{\mathcal{P}}_t \right| \right\rfloor$ (see algorithm in Figure 2).
  Sample $\lfloor \omega\,|\mathcal{P}_t| \rfloor$ from the MB–GNG.
  Substitute randomly selected individuals of $\mathcal{P}_t \setminus \hat{\mathcal{P}}_t$ with the new individuals to produce $\mathcal{P}'_t$.
  $P_{t+1} = \hat{\mathcal{P}}_t \cup \mathcal{P}'_t$.
  $t = t + 1$.
until end condition not met
Determine the set of non–dominated individuals of $\mathcal{P}_t$, $\mathcal{P}^*_t$.
return $\mathcal{P}^*_t$ as the algorithm's solution.

Figure 4: Algorithmic representation of MONEDA.

5.1 Test problems

Most experiments involving MOPs deal with only two or three objective problems. In these experiments we intend to deal with higher dimensional problems since we are interested in assessing MONEDA's scalable behavior. Consequently, we have chosen for our analysis some of the members of the DTLZ family of scalable problems (**?**), in particular, the DTLZ3, DTLZ6 and DTLZ7 problems, and some of the WFG set of scalable problems (**?**), in particular WFG1, WFG2 and WFG6.

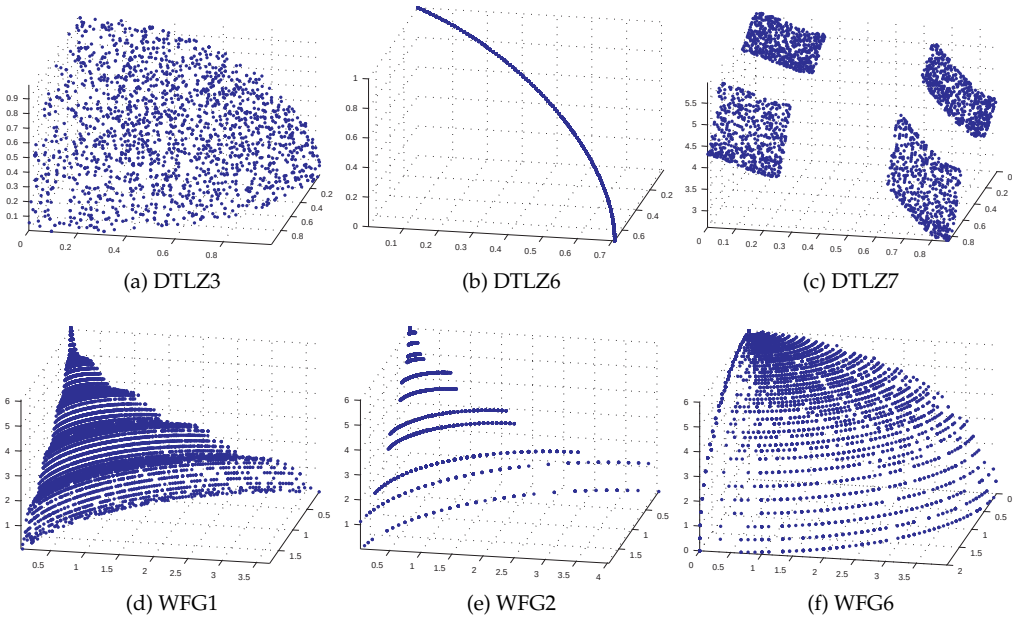
The DTLZ problems were selected for our experiments because of the relative simplicity of their specification, and the existence of an a priori known Pareto–optimal front.

The DTLZ3 problem is a M–objective problem with an n–dimensional decision vector. Its Pareto–optimal front lies on the first orthant of a unit hypersphere (see Figure 5a for a 3–D representation). This problem was introduced to test the ability of a MOEA to converge to the global Pareto–optimal front, since there are $3^{n-M+1} - 1$ suboptimal fronts parallel to the optimal one.

The DTLZ6 problem is based on a simpler problem, in this case the DTLZ5 problem. As in the previous case, suboptimal fronts are also present with the intention of deceiving the optimizer. For a graphical representation see Figure 5b.

On the other hand, the DTLZ7 problem has a Pareto–optimal front that consists of a heavily disconnected set of 2^{M-1} Pareto–optimal regions that test an algorithm's ability to maintain a robust coverage of all optimal regions. Figure 5c illustrates a 3–D graphical representation of the Pareto–optimal front of DTLZ7.

To complement the above, relatively simple problems, we also deal with a selection of the more complex walking fish group (WFG) problem set (**??**). The WFG problem construction tool kit was devised with the aim of providing experimenters with a set of synthetic problems where the problem features were not "hard–wired" in the problem definition. Instead, a target set of characteristics, like bias, multi–modality,



(a) DTLZ3 (b) DTLZ6 (c) DTLZ7

(d) WFG1 (e) WFG2 (f) WFG6

Figure 5: Representation of the Pareto–optimal fronts of DTLZ3, DTLZ6, DTLZ7, WFG1, WFG2 and WFG6 problems configured with three objectives ($M = 3$).

non–separability, non–linearity, etc., can be plugged in and combined as required.

The WFG problem set was devised as a particular instantiation. It consists of nine problems. Of these, we chose WFG1, WFG2 and WFG6 because of their properties and inherent complexities. Although these problems share the same formulation of their Pareto–optimal sets, the corresponding Pareto–optimal fronts each have a different appearance. WFG1 is a separable, uni–modal problem with polynomial and flat bias. Its Pareto–optimal front has a mixed convex geometry, as shown in Figure 5d. WFG2 differs from WFG1 in that it is non–separable, multi–modal and has a disconnected Pareto–optimal front. The properties of its Pareto–optimal front are illustrated in Figure 5e. Finally, WFG6 is non–separable, uni–modal and has a concave Pareto–optimal front (Figure 5f).

In all cases, the complexity of the problems was configured in a progressive fashion, increasing the number of objective space dimensions, specifically, 3, 6, 9 and 12 objective functions.

It should be noted that these are not the only problems that could be used for the type of experiments we carried out (see **?** and **?** for comprehensive reviews). Unfortunately, because of the high computational demands of such experiments and the limited amount of resources available, we were forced to confine the study to fewer problems in order to be able to deal with a larger number of objectives.

5.2 Measuring performance

Assessing the results of a multi–objective optimizer has become a major area of research in the multi–objective community (**?**), because it is essential to compare the results of applying different methods. This is not only an important but also a particularly com-

plex task. It necessarily implies a reduction from an M–dimensional space to a scalar value. In this, as in any dimensionality reduction, valuable information might be lost, leading to invalid conclusions. This situation has been thoroughly described in (**??**).

In practice, there are three fundamental aspects that are taken into account when evaluating the quality of a solution, in particular,

- how close the solutions are to the Pareto–optimal front;
- how much of the Pareto–optimal front is covered by the solutions, and
- how diverse the solutions are in both the decision and objective spaces.

In these experiments, we focused on the first two aspects. Three community–accepted performance indicators were chosen in order to gauge the solutions: the unary additive ϵ–indicator, the hypervolume indicator and the Pareto–optimal front coverage indicator.

5.2.1 Additive ϵ–indicator

The ϵ–indicators (**??**) are a set of performance indicators that rely on the ϵ–dominance concept. These indicators measure how close the local Pareto–optimal front, $\mathcal{PF}_t^*$, is to the global front, $\mathcal{O}^*$.

ϵ–dominance is a relaxed version of the domination relation presented in definition 2. It can be defined in multiplicative and additive terms, but the discussion here is confined to the additive version, as it is the definition employed in this work.

Additive ϵ–dominance is defined as:

Definition 5 *Additive ϵ–dominance relation. For the optimization problem specified in (1), where $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{D}$, $\boldsymbol{x}_1$ is said to additively ϵ–dominate $\boldsymbol{x}_2$ (expressed as $\boldsymbol{x}_1 \preccurlyeq_{\epsilon+} \boldsymbol{x}_2$) iff $f_j(\boldsymbol{x}_1) \leq \epsilon + f_j(\boldsymbol{x}_2)$.*

The additive epsilon indicator, $I_{\epsilon+}(\mathcal{A}, \mathcal{B})$, is a relative indicator that expresses the minimum value of ϵ that is necessary to make a set $\mathcal{A}$ ϵ–dominate a set $\mathcal{B}$, that is,

$$I_{\epsilon+}(\mathcal{A}, \mathcal{B}) = \inf_{\epsilon \in \mathbb{R}} \left\{ \forall \boldsymbol{y} \in \mathcal{B},\ \exists \boldsymbol{x} \in \mathcal{A} \text{ such that } \boldsymbol{x} \preccurlyeq_{\epsilon+} \boldsymbol{y} \right\} . \tag{19}$$

The value of the indicator is to be minimized. $I_{\epsilon+} < 0$ implies that $\mathcal{A}$ strictly dominated $\mathcal{B}$.

To assess the progress of an optimization algorithm, one of the two sets must be substituted by the global Pareto–optimal front, $\mathcal{O}^*$, and the other by the actual Pareto–optimal front, $\mathcal{PF}_t^*$. In practice, however, it is often necessary to generate a discrete version of that set, $\tilde{\mathcal{O}}^*$, by sampling $\mathcal{O}^*$. The correct determination of this set has a direct impact on the accuracy of the indicator.

With this, the indicator can be computed at time $O\left(M \left|\mathcal{A}\right| \left|\mathcal{B}\right|\right)$.

5.2.2 Hypervolume indicator

The hypervolume indicator, $I_{\text{hyp}}(\mathcal{A})$, (**????**) computes the volume of the region, H, delimited by a given set of points, $\mathcal{A}$, and a set of reference points, $\mathcal{N}$.

$$I_{\text{hyp}}\left(\mathcal{A}\right) = \text{volume}\left(\bigcup_{\forall \boldsymbol{x} \in \mathcal{A}; \forall \boldsymbol{y} \in \mathcal{N}} \text{hypercube(a, n)} \right) . \tag{20}$$

Therefore, larger values of the indicator will indicate better solutions.

To measure the absolute performance of an algorithm the reference points should ideally be nadir points. These points are the worst elements of $\mathcal{O}$, or, in other words, the elements of $\mathcal{O}$ that do not dominate any other element. To contrast the relative performance of two sets of solutions, though, one can be used as the reference set. These matters are further detailed in (**??**).

Having $\mathcal{N}$, the computation of the indicator is a non–trivial problem. Indeed, its determination is known to be computationally intensive, thus rendering it unsuitable for problems with many objectives.

A lot of research has focused on improving the computational complexity of this indicator (**?????**). According to the most recent results, the indicator is currently known to be $O(n \log n + n^{M/2})$ (**?**) for more than three objectives ($M > 3$); $O(n \log n)$ for $M = 2, 3$ (**?**).

5.3 Statistical testing

In order to reach substantiated conclusions, we have to do more than just report the descriptive statistics of the performance indicators. To do this, we have to carry out a set of statistical inferences that would support any deductions made from the data.

In our case, the standard statistical inference of interest would assert whether the distribution of the local Pareto–optimal set output by one optimizer is better than another. This cannot be determined unequivocally because we have only a finite sample of the local Pareto–optimal sets output by the algorithms. Therefore, some statistical hypothesis test should be applied to get a measure of how likely the above claim is to be true.

Some statistical hypothesis tests have to be applied to validate the results of different executions. Other authors (see for example (**???**)) have already discussed different frameworks for this purpose.

In our case, for each problem/dimension combination, we performed a Kruskal–Wallis test (**?**) with the indicator values yielded by each algorithm's run. In this context, the null hypothesis of this test is that all algorithms are equally capable of solving the problem. If the null hypothesis was rejected, which was actually the case in all instances of the experiment, the Conover–Inman procedure **?**, pp.288–290 was applied in a pairwise manner to determine whether one algorithm had significantly better results than another. A significance level, α, of 0.05 was used for all the tests. A similar test framework was previously applied for assessing similar experiments (**??**).

There are some possible alternatives, like the use of the Mann–Whitney–Wilcoxon U test (**??**) or the Kolmogorov–Sminoff test (**?**). We decided to apply the above test methodology, as it has been successfully used before in multi–objective experimental contexts. It is also used to compare the performance of the algorithms across problems and number of objectives, as described later in this section.

5.4 Measuring algorithms' computational costs

Besides measuring how good the algorithm solutions are, it is also very important to understand how much computational effort it takes to arrive at those solutions. This effort is expressed in two ways: spatial and temporal. Spatial effort refers to how much storage space (memory, disk, etc.) an algorithm uses during the optimization process. Temporal effort deals with the time that it takes the algorithm to reach the solution. We are concerned with this latter magnitude, as it is the most critical point given the current state of computing technology.

Because of the nature of evolutionary approaches, the traditional methods for esti-

mating the computational complexity of algorithms are no longer suitable. That is why different alternative strategies for assessing the temporal complexity of multi–objective optimizers have been put forward.

The simplest one is to measure the time employed in each independent run and then output a mean execution time. This procedure is sensitive to the uncontrollable influence of concurrent hardware and software processes, like memory swapping, garbage collection, etc., that might interfere with an accurate measurement.

A more common approach is to compute the number of algorithm iterations (the number of generations in the evolutionary case) needed to achieve the results. This method has the advantage of providing a measurement that is repeatable using different hardware and software combinations. On the downside, it does not account for the time taken to complete each iteration. This intra–iteration time is often considerable, and therefore the wrong conclusions could be drawn if it is disregarded.

The third strategy counts the number of evaluations of the objective functions. This method is rooted in real–life engineering problems, where evaluations are usually costly and should be minimized. This approach provides more complete information than the others. Even so, it does not take into account the amount of computation spent on the actual optimization process, which can be the most time-consuming parts.

One way of gaining a better understanding of time complexity is to measure the number of floating–point operations carried out by each algorithm. This approach assumes that all floating–point operations have to do with the optimization process itself. This requirement can be easily met under experimental conditions.

There are a number of profiling tools that are capable of tracking the number of floating–point operations that have taken place as part of a process. For this research, we chose the OProfile program profiling toolkit (**?**).

5.5 Experiment design

Experiments were carried out under the PISA framework (**?**). Algorithm implementations were adapted from the ones provided by their respective authors, with the exception of NSGA–II and SPEA2, which were already distributed as part of the framework, and MOPED and MONEDA, which were implemented from scratch. In all cases, the code was reviewed to ensure its optimality in order to output valid temporal complexity measurements.

A correct selection of each algorithm's initial parameters has a direct impact on the validity of experiments like the ones we are proposing. Normally, initial parameters are selected after a preliminary hand tuning for small–scale problems Because of the high computational demands of the experiments in this study, this adjustment phase has been limited to the three dimensional problems. Some parameters, however, needed to be bound to the dimension of the objective space, M. In such cases an explicit bounding formula was used. The parameters selected for each algorithm are summarized in Table 2. Whenever possible, we have used the algorithm parameters as reported in their corresponding papers. This should ensure the reproducibility and comparability with previously published results.

Experiments were carried out on a 3.4 GHz Intel Quad–core computer with 4 GB of RAM memory running the Linux operating system. Each execution was repeated 30 times in order to output statistically significant results.

Stopping an optimizer is in itself a complex matter. It is usual practice in the evolutionary field to stop experiments after a given number of iterations. This strategy is of no use for this study since it is impossible to predict how much computation is re-

Table 2: Parameters of the algorithms used in the experiments.

Common parameters	
Population size (n_{pop})	$250 \cdot 10^{\frac{M}{3}-1}$
MONEDA	
Number of initial GNG nodes (N_0)	2
Maximum edge age ($\nu_{\max}$)	40
Best node learning rate (ϵ_{b})	0.1
Neighboring nodes learning rate (ϵ_{v})	0.05
Insertion error decrement rate (δ_{I})	0.1
General error decrement rate (δ_{G})	0.1
Accumulated error threshold (ρ)	0.2
Selection percentile (α)	0.3
$\hat{P}_t$ to $N_{\max}$ ratio (γ)	0.5
Substitution percentile (ω)	0.25
RM–MEDA	
Selection portion	0.3
Number of LPCA clusters	$\frac{10}{3}M$
Maximum training steps in LPCA	$\frac{20}{3}M$
Extension rate	0.25
MOPED	
Selection portion	0.3
Sampling parameter (τ)	2
Fitness parameter (α)	0.2

Naïve MIDEA	
Selection percentile (τ)	0.3
Diversity percentile (δ)	15
Number of parents of a variable (κ)	2
Maximum number of clusters	$\lceil 0.5 \lfloor \tau n_{\text{pop}} \rfloor \rceil$
Threshold for the leader algorithm	0.1
MrBOA	
Selection portion (τ)	0.3
Number of parents or a variable	5
Number of mixture components	3
Threshold of leader algorithm	0.3
NSGA–II	
Crossover probability (p_c)	0.7
Distribution index for SBX (η_c)	15
Mutation probability (p_m)	$\frac{1}{n_{\text{pop}}}$
Dist. index for polynomial mut. (η_m)	20
SPEA2	
Crossover probability (p_c)	0.7
Distribution index for SBX (η_c)	15
Mutation probability (p_m)	$\frac{1}{n_{\text{pop}}}$
Dist. index for polynomial mut. (η_m)	20
Ratio of pop. to archive sizes	4 : 1

quired to solve high–dimensional problems. This is why we applied the MGBM multi–objective optimization stopping criterion (**??**). This criterion is a relative, domination–based criterion that was previously developed with these types of applications in mind.

The above performance indicators have to have a reference set of points. The the additive ϵ–indicator uses the Pareto–optimal front, $\mathcal{O}^*$, to carry out their calculations. Similarly, the hypervolume indicator needs a set of nadir points. All this poses a problem when carrying out experiments that deal with many dimensions. This problem is particularly acute in cases requiring the Pareto–optimal front. In such cases, even when an explicit formulation of the front exists, it is computationally unviable to sample well enough to output $\tilde{\mathcal{O}}^*$.

The test problems employed in this paper, with the exception of DTLZ3 and WFG6, suffer from this drawback. In the case of DTLZ3 and WFG6, as their $\mathcal{O}^*$ lie on the first orthant of an hypersphere of radius 1 situated on the origin of coordinates, it is straightforward to determine the distance from any point in the objective space to $\mathcal{O}^*$. However, this feature for calculating the Pareto–optimal front coverage is of no use, as it requires a sampled version of $\mathcal{O}^*$.

To address this issue we have taken an alternative approach, similar to the one used by the purity performance indicator (**??**). A combined set $\mathcal{PF}^+$ is defined as the union of the solutions of the different algorithms across all the experiment executions. $\tilde{\mathcal{O}}^*$ is then determined by extracting the non–dominated elements

$$\boldsymbol{x} \in \tilde{\mathcal{O}}^* \text{ iff } \boldsymbol{x} \in \mathcal{PF}^+ \text{ and } \nexists \boldsymbol{y} \in \mathcal{PF}^+ \text{ such that } \boldsymbol{y} \prec \boldsymbol{x}\,. \tag{21}$$

Although this procedure circumvents the problems of performing a direct sampling of the Pareto–optimal front shape function, special precautions should be taken when interpreting the results. Note that algorithm performance will be measured with regard

to the set of best overall solutions; not against the actual Pareto–optimal front.

A similar method is used to determine the nadir points set used by the hypervolume indicator. In this case $\mathcal{N}$ is computed as

$$\boldsymbol{x} \in \mathcal{N} \text{ iff } \boldsymbol{x} \in \mathcal{PF}^{+} \text{ and } \nexists \boldsymbol{y} \in \mathcal{PF}^{+} \text{ such that } \boldsymbol{x} \prec \boldsymbol{y}\,. \tag{22}$$

5.6 Quality and performance analysis

Figures 6, 7 and 8 show the statistical description of the final results yielded by the algorithm runs when dealing with the DTLZ problems in the form of box plots (**??**) and as tables that summarize the outcome of the above statistical tests. This representation assists in the assessment of the quality and validity of the final solutions of the algorithms. In all cases, the performance was computed using the results of the iterations marked by the stopping criterion.

In the three–dimensional configurations, MONEDA performed similarly to the other algorithms. This was an expected outcome since our MOEDA uses an already existent fitness function and its model–building algorithm is meant to provide a significant advantage in more extreme situations.

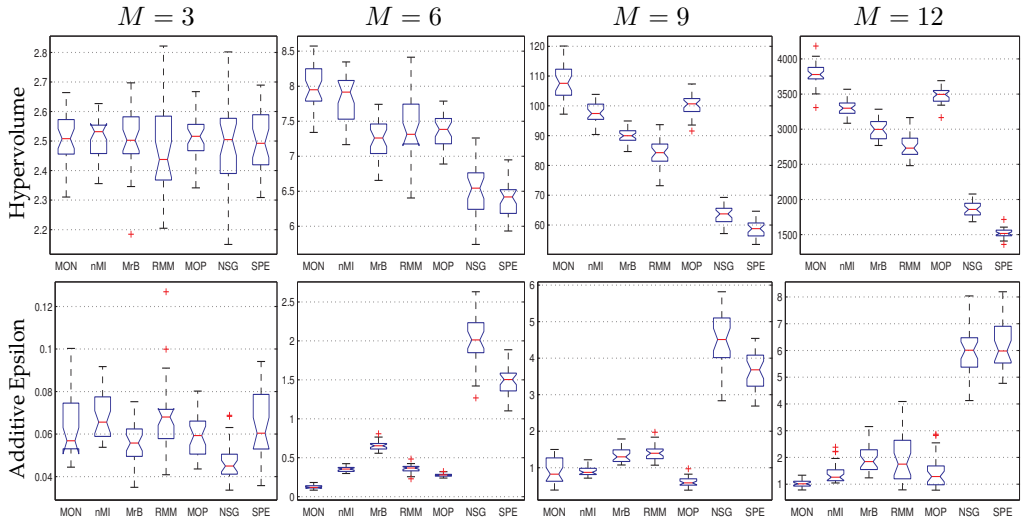
However, as the tests move into higher dimensions, it becomes evident that MONEDA outperforms the other applied optimizers. Not only does MONEDA yield better final results, but also results variance is very low and fewer iterations are required. This means that it performed consistently well across the different runs. The results not only show that MONEDA's solutions are close to the optimal but also that they manage to evenly cover the Pareto–optimal front.

In spite of the encouraging results described here, special care should be taken during analysis. Notice that because of the methodology used for generating the surrogate Pareto–optimal front, $\tilde{\mathcal{O}}^{*}$, the ϵ–indicator is not contrasting the solutions against the true optimal front. This could potentially bias the findings.

In this light, the outstanding results of MONEDA should be interpreted in relative terms. MONEDA's low $I_{\epsilon+}$ values indicate that its solutions were better than most of the solutions of the other algorithms and they, therefore, belong to the joint local Pareto–optimal front, $\tilde{\mathcal{O}}^{*}$. Similarly, although MONEDA has managed to produce better results when compared to the other algorithms, its solutions cannot be said to be close enough to the Pareto–optimal front. The scheme applied to determine $\tilde{\mathcal{O}}^{*}$ may also be the cause of the particularly low variance of MONEDA's results.

Nevertheless, the analysis of the variances of the indicators prompts a most interesting side discussion. On the one hand, the $I_{\epsilon+}$ indicator reports relatively low variances for most algorithms and particularly small for MONEDA. On the other, the spread of the measurements of I_{H} is greater. Again, this is probably caused by the scheme we used to determine $\tilde{\mathcal{O}}^{*}$. The initial conclusion of this is that, for experiments like the ones we are analyzing here, it is of more use to apply performance indicators that do not depend on a known Pareto–optimal front, such as the I_{H} indicator. Anyhow, in spite of the higher variances, MONEDA still yields better and more statistically sound results than the other algorithms.

In spite of the relevant information that can be extracted from the above visual representation, a more formal approach is necessary to confirm the stated results from a statistical point of view. The results of applying the Mann–Whitney test to the outcome of the above experiments is summarized in Tables 6b, 7b and 8b for the DTLZ3, DTLZ6 and DTLZ7 problems, respectively. They verify that MONEDA was able to tackle the problem better than rest of the algorithms in most experiments with 6 objectives and beyond.



(a) Graphical representation of the indicators as box–plot.

(b) Results of the statistical test for the hypervolume indicator.

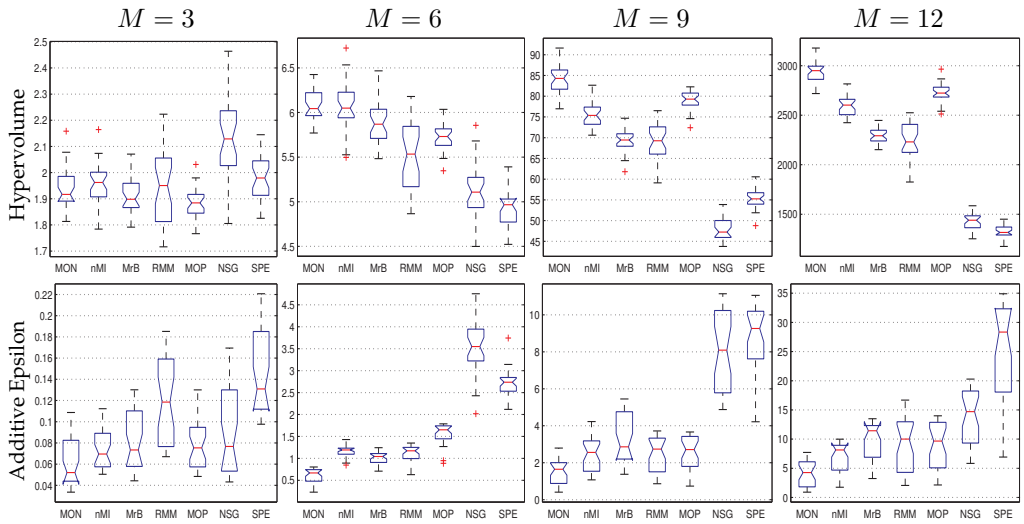
	MON	nMI	MrB	RMM	MOP	NSG	SPE
MONEDA	—	9, 12	6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12
naïve MIDEA		—	6, 9, 12	6, 9, 12	6	6, 9, 12	6, 9, 12
MrBOA			—	9, 12		6, 9, 12	6, 9, 12
RM–MEDA				—		6, 9, 12	6, 9, 12
MOPED					—	6, 9, 12	6, 9, 12
NSGA–II						—	9, 12
SPEA2							—

Figure 6: Summary of the statistical description of the results yielded by the MONEDA (MON), naïve MIDEA (nMI), MrBOA (MrB), RM–MEDA (RMM), MOPED (MOP), NSGA–II (NSG) and SPEA2 (SPE) algorithms when solving the DTLZ3 problem. Figure 6a presents the indicator values obtained after each experiment as box–plots. Table 6b summarizes the outcome of performing the statistical hypothesis tests. The numbers shown are the problem dimension where the test detected a statistically significant better indicator values of the algorithm in each row with respect of those in the columns.

Similar conclusions can be drawn from the experiments involving the WFG1, WFG2 and WFG6 problems. Although, in general terms, these problems pose a bigger challenge to the optimizers, the progress shapes of the algorithms are rather similar to those of the previous problems. Figures 9, 10 and 11 respectively, which depict the statistical properties of the indicator values yielded by each algorithm, illustrate this point.

The results of the WFG problems share the same properties as explained above. Even though the scalar values of the indicators change, the outcome of comparing algorithm performance is more or less the same.

The critical assessment of these results led us to hypothesize that, thanks to its novel treatment of the outliers in the model–building data set, our approach manages to overcome the weaknesses that are an obstacle to the other methods. Although very interesting, the results presented here raise the question of how conditioned they are



(a) Graphical representation of the indicators as box–plots.

(b) Results of the statistical test for the hypervolume indicator.

	MON	nMI	MrB	RMM	MOP	NSG	SPE
MONEDA	—	9, 12	6, 9, 12	6, 9, 12	3, 6, 9, 12	6, 9, 12	6, 9, 12
naïve MIDEA		—	3, 6, 9, 12	6, 9, 12	3, 6	6, 9, 12	6, 9, 12
MrBOA			—	6	6	6, 9, 12	6, 9, 12
RM–MEDA				—		6, 9, 12	6, 9, 12
MOPED					—	6, 9, 12	6, 9, 12
NSGA–II						—	3, 6, 12
SPEA2							—

Figure 7: Statistical description of the results yielded by the algorithms involved in the experiments when solving the DTLZ6 problem. See Figure 6 for an extended description.

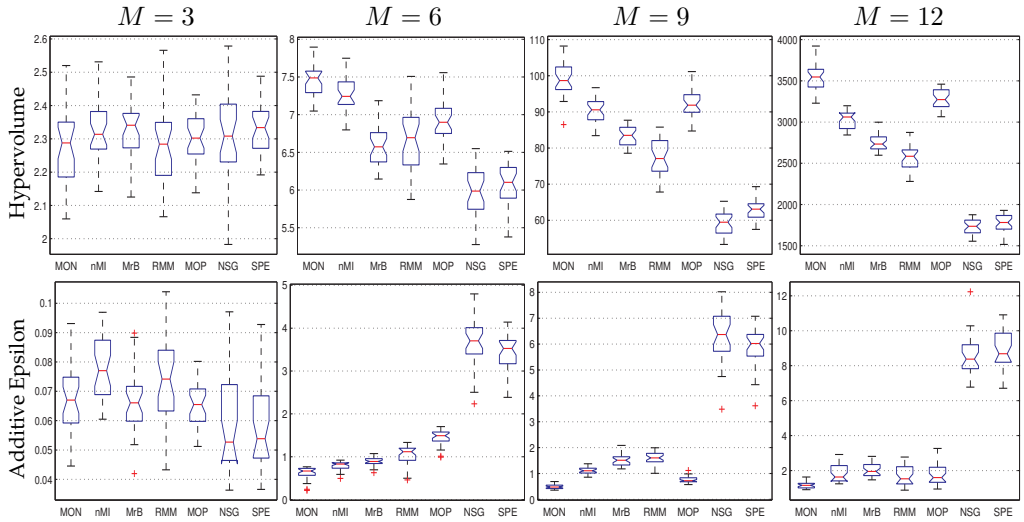
by the particularities of the solved problems. This issue requires further investigation, to understand whether the low dispersion of the error indicators is achieved for the solved problems only, or can be extrapolated to other more complex problems as well.

In any case, one of the most important conclusions of these experiments is that MONEDA has shown itself to be a robust algorithm. Despite having a relatively large number of parameters, MONEDA is capable of dealing with a wide range of problems, each with different characteristics, without having to custom tune its configuration.

It is rather hard to confirm these facts, as it implies cross–examining and comparing the results presented separately. For this reason, we decided to adopt a more integrative representation, as proposed in (**??**). That is, for a given set of algorithms $A_1,\ldots,$ A_K, a set of P test problem instances $\Phi_{1,m},\ldots,\Phi_{P,m}$, configured with m objectives, the function $\delta(\cdot)$ is defined as

$$\delta\left(A_i, A_j, \Phi_{p,m}\right) = \begin{cases} 1 & \text{if } A_i \gg A_j \text{ solving } \Phi_{p,m} \\ 0 & \text{in other case} \end{cases}, \tag{23}$$

where the relation $A_i \gg A_j$ defines whether A_i is significantly better than A_j when solving the problem instance $\Phi_{p,m}$, as computed by the above statistical tests.



(a) Graphical representation of the indicators as box–plots.

(b) Results of the statistical test for the hypervolume indicator.

	MON	nMI	MrB	RMM	MOP	NSG	SPE
MONEDA	—	6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12
naïve MIDEA		—	6, 9, 12	6, 9, 12	6	6, 9, 12	6, 9, 12
MrBOA			—	3, 9, 12		6, 9, 12	6, 9, 12
RM–MEDA				—		6, 9, 12	6, 9, 12
MOPED					—	6, 9, 12	6, 9, 12
NSGA–II						—	
SPEA2							—

Figure 8: Statistical description of the results yielded by the algorithms involved in the experiments when solving the DTLZ7 problem. See Figure 6 for an extended description.

Relying on $\delta(\cdot)$, the performance index $P_{p,m}(A_i)$ of a given algorithm A_i when solving $\Phi_{p,m}$ is then computed as

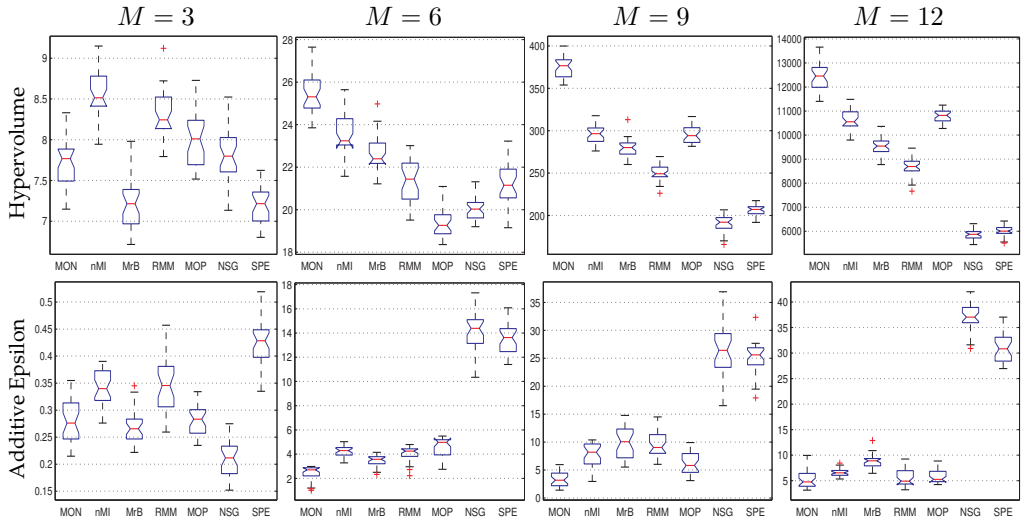
$$P_{p,m}\left(A_i\right) = \sum_{j=1;j\neq i}^{K} \delta\left(A_i, A_j, \Phi_{p,m}\right). \tag{24}$$

This index intends to summarize the performance of each algorithm with regard to its peers.

Fig. 12 exhibits the results of computing the performance indexes. Fig. 12a represents the mean performance indexes yielded by each algorithm when solving each problem in all of its configured objective dimensions,

$$\bar{P}_p\left(A_i\right) = \frac{1}{|\mathcal{M}|} \sum_{m\in\mathcal{M}} P_{p,m}\left(A_i\right). \tag{25}$$

It is worth noticing that MONEDA has better overall results with respect to the other algorithms. The relatively poor overall performance of the randomized leader



(a) Graphical representation of the indicators as box–plots.

(b) Results of the statistical test for the hypervolume indicator.

	MON	nMI	MrB	RMM	MOP	NSG	SPE
MONEDA	—	6, 9, 12	3, 6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12	3, 6, 9, 12
naïve MIDEA		—	3, 6, 9, 12	3, 6, 9, 12	3, 6	3, 6, 9, 12	3, 6, 9, 12
MrBOA			—	6, 9, 12	6	6, 9, 12	6, 9, 12
RM–MEDA				—	3, 6	3, 6, 9, 12	3, 9, 12
MOPED					—	9, 12	3, 9, 12
NSGA–II						—	3
SPEA2							—

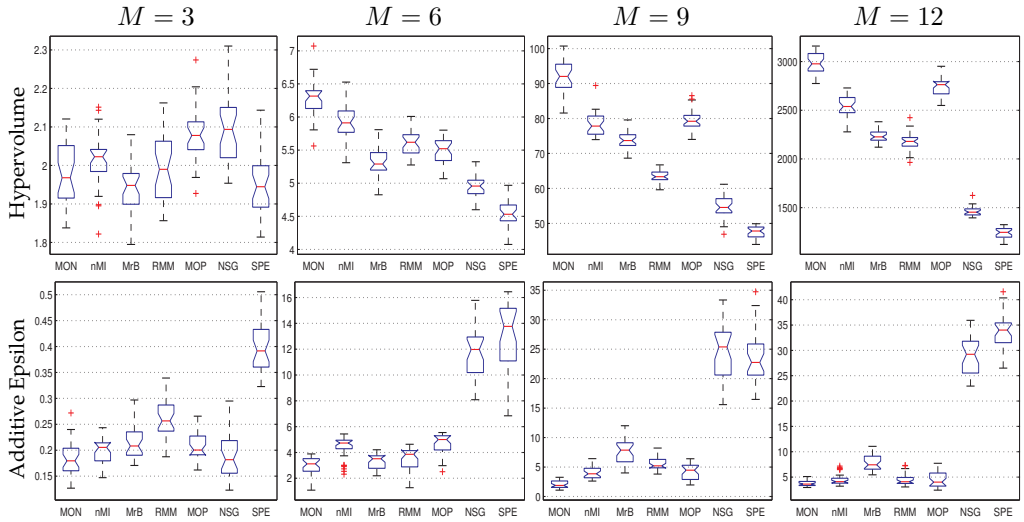
Figure 9: Statistical description of the results yielded by the algorithms involved in the experiments when solving the WFG1 problem. See Figure 6 for an extended description.

and the k–means algorithms for some problems, like WFG5 and WFG7, and WFG8 and WFG9, respectively, was quite unexpected. A possible hypothesis is that the three objective problems are biasing these results, where there are dramatic differences compared to the results for the other dimensions.

This situation is clarified in Fig. 12b, which presents the mean values of the index computed for each dimension,

$$\bar{P_m}\left(A_i\right) = \frac{1}{P}\sum_{p=1}^{P} P_{p,m}\left(A_i\right) . \tag{26}$$

In this case, it can be corroborated that there is no substantial difference in the results produced by the different algorithms in the three objective cases, as their indexes have more evenly shared values. It is also noticeable that CMA–ES seems to outperform all the other algorithms in all problems in this dimension. This panorama changes when inspecting the higher dimensionality results (in the objective function space). In those cases the least statistically robust algorithms tend to perform comparatively better, with the exception of Bayesian networks that seem to improve as the number of dimensions increases, but, of course, at the expense of a great computational cost.



(a) Graphical representation of the indicators as box–plots.

(b) Results of the statistical test for the hypervolume indicator.

	MON	nMI	MrB	RMM	MOP	NSG	SPE
MONEDA	—	6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12
naïve MIDEA		—	3, 6, 9, 12	6, 9, 12	6	6, 9, 12	3, 6, 9, 12
MrBOA			—	9, 12		6, 9, 12	6, 9, 12
RM–MEDA				—		6, 9, 12	6, 9, 12
MOPED					—	6, 9, 12	3, 6, 9, 12
NSGA–II						—	3, 6, 9, 12
SPEA2							—

Figure 10: Statistical description of the results yielded by the algorithms involved in the experiments when solving the WFG2 problem. See Figure 6 for an extended description.

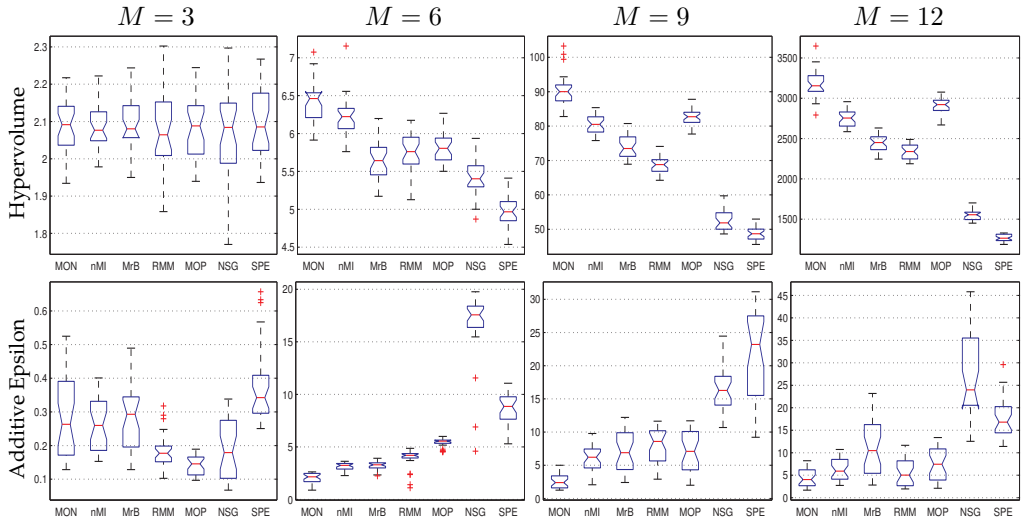
It is worthwhile analyzing the performance of MB–GNG. In most cases, MB–GNG outperformed the other algorithms in higher dimensionality. This corroborates the results that we presented in previous works (**??**). This outcome can be attributed to the fact that MB–GNG is the only algorithm that has so far been specially devised for the model–building problem.

5.7 Computational cost of the algorithms

One of our main concerns when comparing the different algorithms was their computational requirements. One simple and illustrative way of doing this is to plot the progress of the different algorithms when solving each problem, as already argued in Section 5.4.

Figure 13 summarizes the average number of iterations for each algorithm, the mean floating–point CPU operations per iteration and the mean total floating–point CPU operations used by the algorithms when solving the DTLZ problems. Figure 14 contains the same analysis for the WFG problems.

This set of measurements reinforces our earlier conclusions. The figures suggest that, besides requiring relatively few iterations to converge, it also performs fewer op-



(a) Graphical representation of the indicators as box–plots.

(b) Results of the statistical test for the hypervolume indicator.

	MON	nMI	MrB	RMM	MOP	NSG	SPE
MONEDA	—	6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12	6, 9, 12
naïve MIDEA		—	6, 9, 12	6, 9, 12	6	6, 9, 12	6, 9, 12
MrBOA			—	9, 12		6, 9, 12	6, 9, 12
RM–MEDA				—		6, 9, 12	6, 9, 12
MOPED					—	6, 9, 12	6, 9, 12
NSGA–II						—	6, 9, 12
SPEA2							—

Figure 11: Statistical description of the results yielded by the algorithms involved in the experiments when solving the WFG6 problem. See Figure 6 for an extended description.

erations in every iteration.

The cases of MrBOA and RM–MEDA are very illustrative. Although they require fewer iterations, their mean CPU operations per iteration are the highest, as are their total number of CPU operations, too. This makes the case for not measuring the number of iterations or function evaluations only in this class of experiments.

Another interesting phenomenon is the relatively low increase in the number of iterations when moving from 6 to 9 objectives. This behavior is shared across most algorithms and problems. In our opinion it can be attributed to the relatively large size of the population used.

Also noticeable is the (presumably) exponential increase in the number of iterations and amount of CPU consumption as the problem complexity grows. This means that future algorithms should take into account this problem and at least try to curb this growth.

5.8 Commentaries about the results

It can be argued that, as in any experimental comparison, the parameters of the different algorithms have strongly biased the results. For example, it could be hypothesized

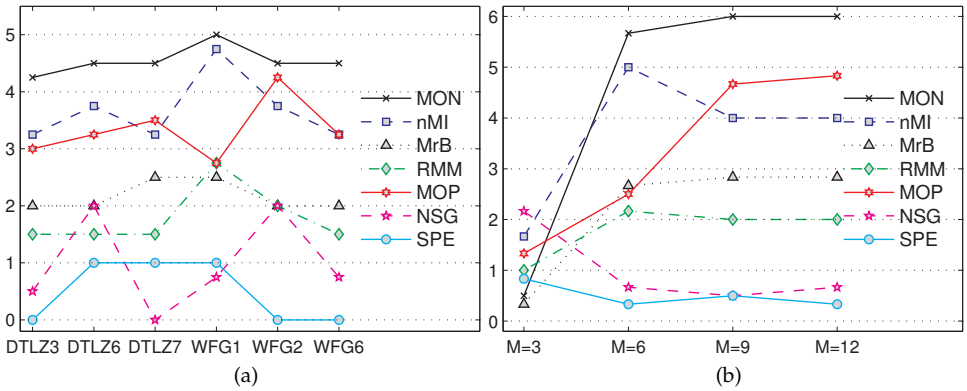


Figure 12: Mean values of the performance index across the different problems, $\bar{P}_p$ (), (Fig. 12a) and objective space dimensions, $\bar{P}_m$ (Fig. 12b).

that NSGA–II and SPEA2, with much larger population sizes, would probably yield better approximations to the Pareto–optimal fronts, albeit at a higher computational expense.

In any case, the most remarkable conclusion, when assembling the results listed in Sections 5.6 and 5.7, is that MONEDA is capable of consistently producing similar or better results with regard to similar approaches at a lower computational cost. In its turn, this improvement can only be attributed to the introduction of a novel model–building algorithms specially designed for the purpose, since other algorithm properties, like, for example, the fitness assignment, were unchanged.

6 Final remarks and salient issues

In this work we have dealt with an open issue of current multi–objective optimization estimation of distribution algorithms: the model–building problem. We argued that current model–building approaches, which are based on off–the–shelf machine learning methods, do not meet the task requirements. In particular, they do not correctly handle outliers; they suffer from population diversity loss, and they expend too many resources on building the model.

To test our arguments we introduced a novel evolutionary algorithm called the multi–objective optimization neural estimation of distribution algorithm (MONEDA). MONEDA puts forward an innovative neural network–based scheme for model–building. In particular, a modified GNG neural network is applied. This model–building algorithm addresses two theoretical and practical issues not taken into account by earlier approaches.

MONEDA's behavior has been assessed on a set of well–known community–accepted problems with a progressive increase in the number of objectives. Its results have been compared against a range of state–of–the–art algorithms.

The experimental results show that MONEDA performs similarly to other approaches in problems with relatively few dimensions. However, as the problem complexity scales up, MONEDA outperforms the other algorithms in terms of the quality of the solutions and their computational complexity.

However, there are many issues that remain open. For example, the algorithm's

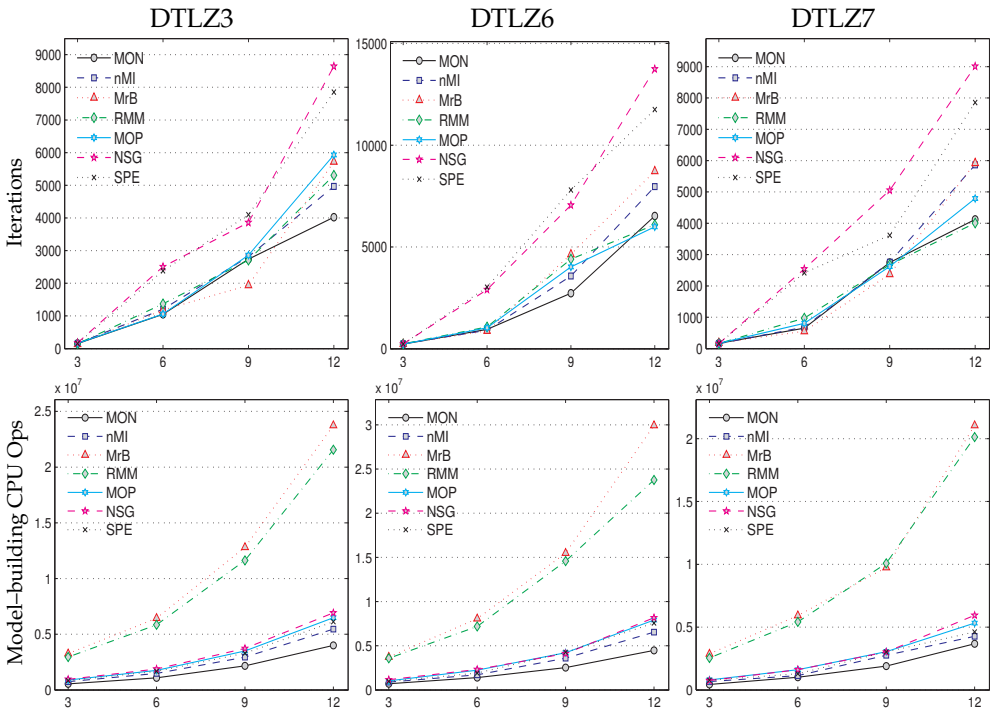


Figure 13: Comparative analysis of the computational cost of the algorithms under study when dealing with the DTLZ3, DTLZ6 and DTLZ7 problems. For each algorithm/problem/number of objectives combination the mean of the cost indicators yielded by each of the 30 runs is computed. The first row represents the mean number of iterations used by each algorithm, while the second, the mean of the intra–iteration floating–point CPU operations used for model–building.

sensitivity to its parameters must be explored in depth. One of the main drawbacks of MONEDA is its rather large number of free parameters. Improvements should target building less parameterized algorithms.

It is equally important to grasp a more extensive range of test problems. These experiments, although very costly in terms of computational resources, are essential for understanding the reach of the improvements suggested here.

Similarly, there is plenty of room for improvement in MONEDA. Different combinations of selection and replacement schemes would probably produce better results. Similarly, there are different fitness assignment strategies, like the methods employed by SPEA2 or PAES, methods based on relaxed forms of Pareto dominance and methods based on performance indicators that should be contrasted in order to determine their suitability. Other interesting lines of research are the reuse of models across iterations and the fusion of the fitness assignment and the model–building processes into a combined process that would be less computationally demanding. In this paper we have focused on improving the model-building algorithm, but, even so, we acknowledge that there are further improvements to be made to fitness assignment after introducing these alternatives.

Beyond the successful outcome of experiments, the most important consequence

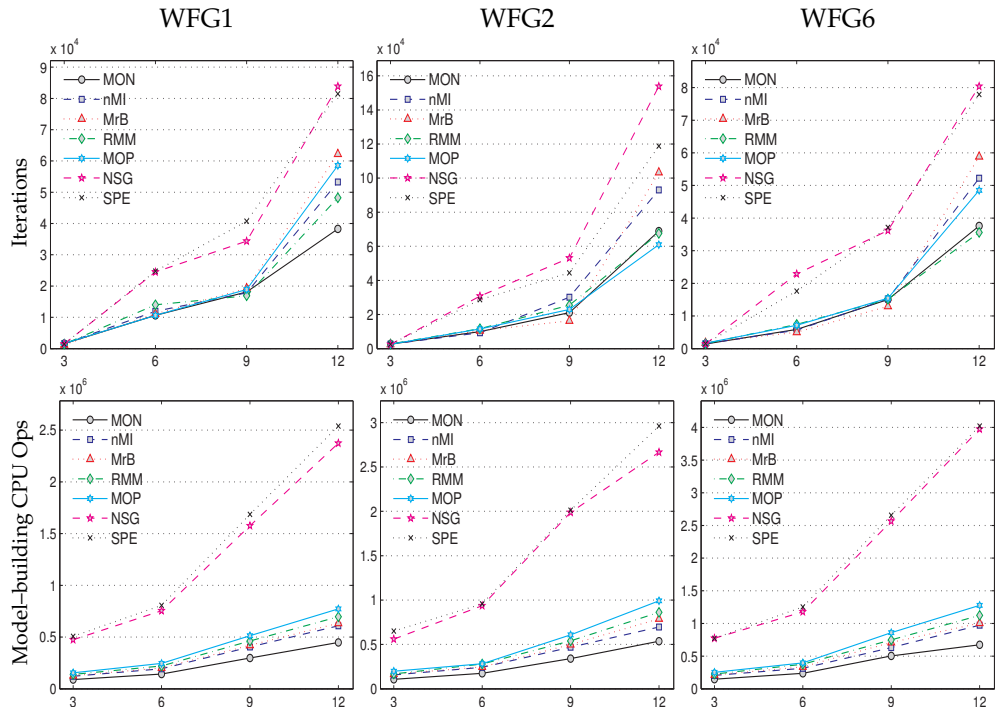


Figure 14: Comparative analysis of the computational complexity of the algorithms under study when dealing with the WFG1, WFG2 and WFG6 problems. For each algorithm/problem/number of objectives combination the mean of the cost indicators yielded by each of the 30 runs is computed. The first row represents the mean number of iterations used by each algorithm, while the second, the mean of the intra–iteration floating–point CPU operations used for model–building.

of this work is that we have exposed a previously overlooked issue in the EDA field. To the best of our knowledge, ours is the first model–building analysis and the proposal. Therefore, this research would be most valuable if it inspires a set of new approaches to the model–building issue. A more exhaustive review of suitable machine learning methods taking the considerations put forward into account would probably yield even better model builders. Perhaps even new methods should be synthesized in order to properly address this task.

Although we have focused on the multi–objective case, the discussed model–building issue can also be extended to single–objective EDAs. In cases where the optimizer must yield more than one optimal solution, like multi–modal optimization problems, the model–building issue should also be manifest. The points that we made about the incorrect treatment of outliers are not out of place here either. These matters, however, open another line of research, which is beyond the scope of this paper.

Acknowledgments

The authors are very grateful to the anonymous reviewers for their insightful comments. The source codes used in the preparation of this paper and the resulting data can be obtained at `http://www.giaa.inf.uc3m.es/miembros/lmarti/moneda`.

This work was supported by projects CICYT TIN2008–06742–C02–02/TSI, CICYT TEC2008–06732–C02–02/TEC, SINPROB and CAM MADRINET S–0505/TIC/0255.