Chapter 20
On the Computational Properties of the Multi-Objective Neural Estimation of Distribution Algorithm

Luis Martí, Jesús García, Antonio Berlanga, and José M. Molina

Abstract. This paper explores the behavior of the multi–objective neural EDA (MONEDA) in terms of its computational requirements it demands and assesses how it scales when dealing with multi–objective optimization problems with relatively large amounts of objectives. In order to properly comprehend these matters other MOEDAs and MOEAs are included in the analysis. The experiments performed tested the ability of each approach to scalably solve many–objective optimization problems. The fundamental result obtained is that MONEDA is not only yields similar or better solutions when compared with other approaches but also does it with at a lower computational cost.

Keywords: Multi–objective Optimization, Computational Complexity, Multi–objective Optimization Evolutionary Algorithms, Estimation of Distribution Algorithms (EDAs).

20.1 Introduction

Multi–objective optimization problems (MOPs) [8] are one of the main research topics of the nature–inspired and evolutionary computation communities. In those problems the optimizer must find a set of feasible solutions that jointly minimize (or maximize) the values of two or more objective functions subject to a set of restrictions. Multi–objective optimization evolutionary algorithms (MOEAs) [3, 5] have been successful in addressing such problems. This can be mainly attributed to their parallel global search and non–assumption of any particular shape of the underlying fitness landscape.

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

Although MOEAs have proved themselves as a satisfactory approach, it has been shown that they dramatically suffer from the curse of dimensionality when addressing MOPs with a relatively large amount of objective functions, known as many–objective optimization problems. A series of experimental studies, like [10, 21] and [5] (pp. 414–419), among others, have shown that there is an exponential dependence between the dimension of the objective space and the amount of resources required to correctly solve the problem.

One of the possible ways of addressing this issue is to employ more efficient evolutionary approaches such as estimation of distribution algorithms (EDAs) [12]. EDAs replace the application of evolutionary operators. Instead they create an statistical model of the fittest elements of the population in an operation known as *model–building process*. This model is then sampled to produce new elements. The extension of EDAs to the multi–objective domain has lead to what can be denominated multi–objective EDAs (MOEDAs) [20]. In spite of its promising properties, MOEDAs have failed to yield a substantial improvement regarding standard EAs when solving many–objective problems [16].

The modification of the model–building algorithm is one of the ways of achieving a substantial progress in this matter. Most model–building schemes used so far by EDAs use off–the–shelf machine learning methods. However, the model–building problem has specific requirements that those methods do not meet and even avoid. In particular, in those algorithms, outliers are treated as invalid data, where in the model building problem they represent newly discovered regions of the search space. Similarly, an excess of resources is spent in finding the optimal size of the model.

The multi–objective neural estimation of distribution algorithm (MONEDA) [17, 15] have been proposed with the aim of effectively dealing with many–objective optimization problems. MONEDA benefits from the simpler algorithmics of estimation of distribution algorithms (EDAs) while improving their handling of complex multi–objective problems. In particular, MONEDA introduces a novel model–building algorithm that addresses the particular requirements of MOPs.

The advantages of MONEDA in terms of optimization efficiency with regard to other MOEDAs and MOEAs have been experimentally established [17, 15]. However, one very important area remains not properly dealt with: the computational characteristics of MONEDA. MONEDA was devised with high–dimensional MOPs in mind therefore it was supposed to not only to be able to find adequate solutions in those situations but to be capable to scale its performance and making feasible to address MOPs with many objectives. Previous experiments produced some evidences that pointed out that MONEDA had a better computational properties when compared to other similar preexisting approaches, but there is not an exhaustive study on the subject.

The leitmotif of this paper is to explore the behavior of MONEDA in terms of its computational requirements and to assess how that quantity scales when dealing with problems with relatively large amounts of objectives. In order to properly comprehend this matters other MOEDAs and MOEAs will be included in the analysis.

The main contribution of this paper is to experimentally establish the advantages of MONEDA with regard to similar methods not only in terms of optimization efficiently but also in terms of computational requirements.

The subsequent parts of this paper first introduce some fundamental concepts needed for the rest of the discussion. After that, MONEDA is briefly discussed, putting it in the context of similar approaches. Following this the particularities of measuring the complexity of a MOEA is described, analyzing different approaches. Afterwards the experiments that have been carried out are described and their results thoroughly commented. As a conclusion, some final remarks and lines of future work are put forward.

20.2 Theoretical Background

A multi–objective optimization problem (MOP) can be expressed as

Definition 1 (Multi–objective Optimization Problem).

$$\left.\begin{array}{l}\text{minimize } \overrightarrow{F}(\overrightarrow{x}) = \langle f_1(\overrightarrow{x}),\ldots,f_M(\overrightarrow{x})\rangle\,,\\ \text{subject to } c_1(\overrightarrow{x}),\ldots,c_C(\overrightarrow{x}) \leq 0\,,\\ \qquad\qquad d_1(\overrightarrow{x}),\ldots,d_D(\overrightarrow{x}) = 0\,,\\ \qquad\qquad \text{with } \overrightarrow{x} \in \mathcal{D}\,,\end{array}\right\} \tag{20.1}$$

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

In general terms, this class of problems does not have a unique optimal solution. Instead an algorithm solving the problem defined in (20.1) should produce a set containing equally good trade–off optimal solutions. The optimality of a set of solutions can be defined relying on the so called *Pareto dominance relation* [19]:

Definition 2 (Pareto Dominance Relation). *For the optimization problem specified in (20.1) and having $\overrightarrow{x}_1, \overrightarrow{x}_2 \in \mathcal{D}$. $\overrightarrow{x}_1$ is said to dominate $\overrightarrow{x}_2$ (expressed as $\overrightarrow{x}_1 \prec \overrightarrow{x}_2$) iff $\forall f_j$, $f_j(\overrightarrow{x}_1) \leq f_j(\overrightarrow{x}_2)$ and $\exists f_i$ such that $f_i(\overrightarrow{x}_1) < f_i(\overrightarrow{x}_2)$.*

The solution of (20.1) is a subset of $\mathcal{D}$ that contains elements are not dominated by other elements of $\mathcal{D}$.

Definition 3 (Pareto–optimal Set). *The solution of problem (20.1) is the set $\mathcal{D}^*$ such that $\mathcal{D}^* \subseteq \mathcal{D}$ and $\forall \overrightarrow{x}_1 \in \mathcal{D}^* \nexists \overrightarrow{x}_2$ that $\overrightarrow{x}_2 \prec \overrightarrow{x}_1$.*

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

Finding the explicit formulation of $\mathcal{D}^*$ is often impossible. Generally, an algorithm solving (20.1) 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 local Pareto–optimal front.

Although MOPs have been addressed with a variety of methods, evolutionary algorithms (EAs) have proved themselves as a competent approach from both theoretical and practical points of view. This fact 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, it is believed they perform consistently well across various types of problems.

Estimation of distribution algorithms (EDAs), like EAs, are population–based optimization algorithms. However, in EDAs, the step where the evolutionary operators are applied is substituted by construction of a statistical model of the most promising subset of the population. This model is then sampled to produce new individuals that are merged with the original population following a given substitution policy. The introduction of machine learning techniques implies that these new algorithms lose the biological plausibility of their predecessors. In spite of this, they gain the capacity of scalably solving many challenging problems, significantly outperforming standard EAs and other optimization techniques. Multi–objective optimization EDAs (MOEDAs) are the extensions of EDAs to the multi–objective domain. Most of MOEDAs are a modification of existing EDAs whose fitness assignment strategy is substituted by one of the commonly used by MOEAs.

20.3 Multi–objective Neural EDA

The multi–objective neural EDA (MONEDA) is a MOEDA that uses a modified growing neural gas (MB–GNG) network as its model–building algorithm. The MB–GNG network is a custom–made model–building algorithm devised to cope with the specifications of the model–building task (see [15] for details).

20.3.1 Model–Building with Growing Neural Gas

Clustering algorithms have been used as part of the model–building algorithms of EDAs and MOEDAs. However, as we discussed in the previous section a custom–made algorithm might be one of the ways of achieving a significant improvement in this field.

As a foundation for our proposal we have chosen the growing neural gas (GNG) network [9]. GNG networks are intrinsic self–organizing neural networks based on the neural gas [18] model. It creates an ordered topology of inputs 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.

Our model building GNG (MB–GNG) is an extension of the original (unsupervised) GNG. 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 $C = \{c_1, \ldots, c_{N^*}\}$, with $N_0 \leq N^* \leq N_{\max}$. Here N_0 and $N_{\max}$ represent initial and maximal amount of nodes in the network.

A node c_i consists of a center, $\vec{\mu}_i$, deviations , $\vec{\sigma}_i$, 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, $\nu_{i,j}$.

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 $\vec{x}$ is randomly selected from the training data set. Then two nodes are selected for being the closest ones to $\vec{x}$. The *best–matching node*, c_b, is the closest node to $\vec{x}$. Consequently, the *second best–matching node*, $c_{b'}$, is determined as the second closest node to $\vec{x}$.

If $c_{b'}$ is not a neighbor of c_b then a new edge is established between them $\mathcal{V}_b = \mathcal{V}_b \cup \{c_{b'}\}$ with zero age, $\nu_{b,b'} = 0$. If, on the other case, $c_{b'} \in \mathcal{V}_b$ the age of the corresponding edge is reset $\nu_{b,b'} = 0$.

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

A clustering error is then added to the best–matching node error accumulator, ξ_b. After that, learning takes place in the best–matching node and its neighbors with rates ϵ_{best} and ϵ_{vic}, respectively. For c_b adaptation follows the rule originally used by GNG,

$$\Delta\vec{\mu}_b = \epsilon_{\text{best}}\left(\vec{x} - \vec{\mu}_b\right). \tag{20.2}$$

However for c_b's neighbors a cluster repulsion term [23] is added to the original formulation, that is, $\forall c_v \in \mathcal{V}_b$,

$$\Delta\vec{\mu}_v = \epsilon_{\text{vic}}\left(\vec{x} - \vec{\mu}_v\right) + \beta \exp\left(-\frac{\left\|\vec{\mu}_v - \vec{\mu}_b\right\|}{\zeta}\right)\frac{\sum_{c_u \in \mathcal{V}_b}\left\|\vec{\mu}_u - \vec{\mu}_b\right\|}{|\mathcal{V}_b|}\frac{\left(\vec{\mu}_v - \vec{\mu}_b\right)}{\left\|\vec{\mu}_v - \vec{\mu}_b\right\|}, \tag{20.3}$$

Here β is an integral multiplier that defines the amplitude of the repulsive force while ζ controls the weakening rate of the repulsive force with respect to the distance between the nodes' centers. We have set them to $\beta = 2$ and $\zeta = 0.1$ as suggested in [22].

After a given amount of iterations a new node is inserted to the network. First, the node with largest error, c_e, is selected the node. Then the worst node among its neighbors, $c_{e'}$, is located. N^* is incremented and the new node, c_{N^*}, is inserted equally distant from the two nodes. The edge between c_e and $c_{e'}$ is removed and two

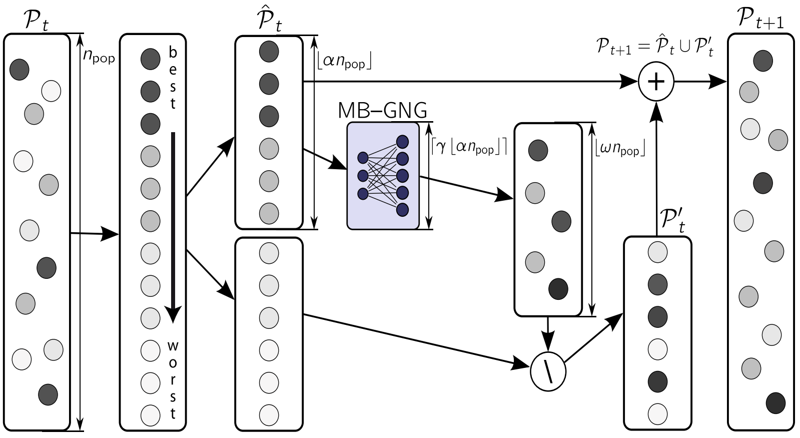

$\mathcal{P}_t$
n_{pop}
best
worst
$\hat{\mathcal{P}}_t$
$\lfloor \alpha n_{\text{pop}} \rfloor$
MB–GNG
$\lceil \gamma \lfloor \alpha n_{\text{pop}} \rfloor \rceil$
$\lfloor \omega n_{\text{pop}} \rfloor$
$\mathcal{P}_{t+1} = \hat{\mathcal{P}}_t \cup \mathcal{P}'_t$
$\mathcal{P}'_t$
$\mathcal{P}_{t+1}$

Fig. 20.1. Diagram representation of the MONEDA algorithm.

new edges connecting c_{N^*} with c_e and $c_{e'}$ are created. Finally, the errors of all nodes are decreased by a factor δ_{G},

$$\xi_i = \delta_{\text{G}} \xi_i, \; i = 1, .., N^*. \tag{20.4}$$

After training has ended the deviations, $\overrightarrow{\sigma}_i$, of the nodes must be computed. For this task the unbiased estimator of the deviations is employed.

20.3.2 MONEDA Algorithmics

MONEDA shares its overall algorithm workflow with other EDAs (see Figure 20.1). It maintains a population of individuals, $\mathcal{P}_t$, with t as the current iteration. It starts from a random initial population $\mathcal{P}_0$ of n_{pop} individuals. It then proceeds to sort the individuals. The NSGA–II non–dominated sorting [5] was the scheme selected for fitness assignment. It was chosen because of its proven effectiveness and its relative low computational cost.

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

A MB–GNG network is then trained using $\hat{\mathcal{P}}_t$ as training data set. In order to have a controlled relation between size of $\hat{\mathcal{P}}_t$ and the maximum size of the network, $N_{\max}$, these two sizes are bound as $N_{\max} = \left\lceil \gamma \left| \hat{\mathcal{P}}_t \right| \right\rceil$. The resulting Gaussian kernels are sampled to produce an amount $\lfloor \omega |\mathcal{P}_t| \rfloor$ of new individuals. Each one of these individuals substitute a randomly selected ones from the section of the population not used for model–building $\mathcal{P}_t \setminus \hat{\mathcal{P}}_t$. The set obtained is then united with best elements, $\hat{\mathcal{P}}_t$, to form the population of the next iteration $\mathcal{P}_t$.

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

20.4 Measuring Complexity

The assessment of an optimizer performance is one of the most vibrant areas of research in the MOEA context. Naturally, most of the efforts have been directed towards the determination of how close are the set of solutions to the Pareto–optimal front.

However, besides determining how good are the solutions obtained from the algorithms it is also very important to understand how big is the computational effort required to reach those solutions. This effort is expressed in two ways: spatial and temporal. The first refers to the amount of storage space (memory, disk, etc.) used by an algorithm during the optimization process. The second deals with the time consumed by the algorithm in order to reach the solution. This last quantity is the one of interest in this work as it is the most critical given the current state of computing technology.

Traditionally there have been three main approaches for assessing the computational cost of multi–objective optimizers. The simplest one is to measure the time taken by each independent run and then obtaining 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 determine the number of algorithm iterations needed for producing the results. This method has de advantage of providing a measurement that is repeatable using different combinations of hardware and software. On the downside, it does not account for the time consumed running each iteration. This intra–iteration time is often considerable, therefore disregarding it may lead to improper conclusions.

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 me minimized. This approach, albeit it provides a more complete information than the previous one, it does not takes into account the amount of computation dedicated to the optimization process itself which can be the most time demanding parts.

An alternative approach that might yield a better understanding on the time complexity is to measure the amount of floating–point CPU operations carried out by each algorithm. This approach assumes that all floating–point operations have to do with the optimization process itself. This is something easily achievable under experimental situations.

There are 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 work we have chosen the OProfile program profiling toolkit [13].

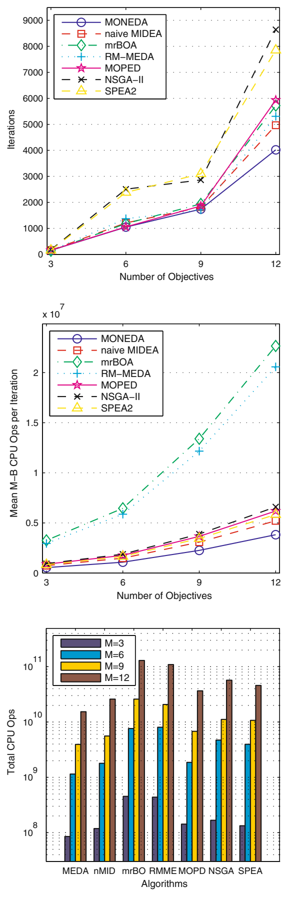
20.5 Experiments

The experiments consist of the application of a set of known and competent evolutionary multi–objective optimizers and MONEDA to some community–accepted

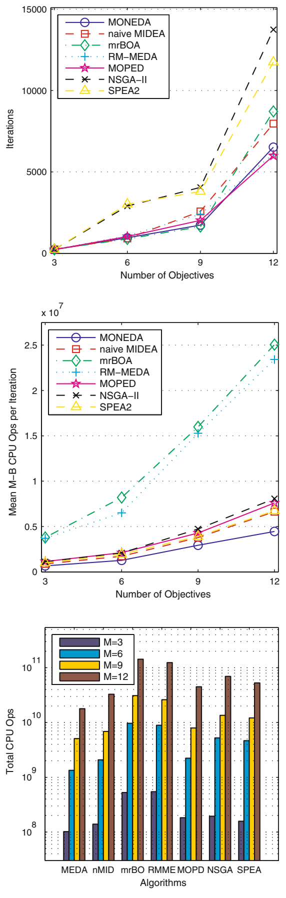
Table 20.1. Hypervolume indicator values (and deviations) for each algorithm solving the DTLZ3, DTLZ6 and DTLZ7 problems.

	$M = 3$	$M = 6$	$M = 9$	$M = 12$
DTLZ3				
MONEDA	2.46(0.10)	**8.09(0.27)**	**107.2(5.08)**	$\mathbf{3.82\times10^3}$**(195.9)**
naïve MIDEA	2.53(0.08)	7.89(0.29)	97.8(3.59)	3.28×10^3(129.8)
mrBOA	**2.54(0.09)**	7.17(0.23)	90.4(3.32)	3.00×10^3(109.0)
RM–MEDA	2.52(0.15)	7.33(0.37)	83.9(4.55)	2.81×10^3(172.7)
MOPED	2.50(0.11)	7.34(0.24)	100.7(2.86)	3.52×10^3(122.9)
NSGA–II	2.49(0.13)	6.34(0.39)	64.5(2.76)	1.84×10^3(96.07)
SPEA2	2.50(0.11)	6.03(0.29)	58.6(2.84)	1.52×10^3(75.01)
DTLZ6				
MONEDA	2.05(0.10)	6.83(0.27)	**83.4(4.46)**	$\mathbf{3.01\times10^3}$**(139.5)**
naïve MIDEA	1.89(0.06)	**7.32(0.30)**	78.3(2.36)	2.52×10^3(117.8)
mrBOA	**2.24(0.09)**	6.05(0.19)	75.3(2.61)	2.28×10^3(125.2)
RM–MEDA	1.94(0.09)	5.69(0.38)	66.1(3.61)	2.18×10^3(156.4)
MOPED	1.94(0.06)	5.84(0.18)	81.3(2.75)	2.66×10^3(114.6)
NSGA–II	2.12(0.11)	5.21(0.31)	49.2(2.49)	1.38×10^3(84.43)
SPEA2	1.92(0.07)	4.54(0.22)	49.6(2.10)	1.20×10^3(64.30)
DTLZ7				
MONEDA	**2.43(0.06)**	**7.80(0.37)**	**103.3(5.17)**	$\mathbf{3.65\times10^3}$**(170.1)**
naïve MIDEA	2.39(0.08)	7.59(0.23)	94.2(2.97)	3.14×10^3(123.6)
mrBOA	2.38(0.10)	6.96(0.28)	85.4(3.21)	2.83×10^3(98.4)
RM–MEDA	2.39(0.11)	6.92(0.37)	79.5(5.35)	2.73×10^3(156.0)
MOPED	2.35(0.07)	7.01(0.26)	94.6(3.14)	3.35×10^3(125.4)
NSGA–II	2.38(0.14)	6.23(0.33)	60.3(3.56)	1.77×10^3(100.6)
SPEA2	2.38(0.11)	5.77(0.18)	56.5(2.88)	1.45×10^3(56.4)

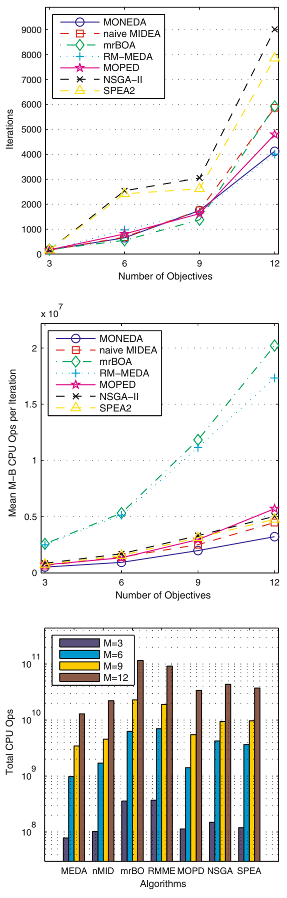
test problems. The algorithms applied are naïve MIDEA [2], mrBOA [1], RM–MEDA [24], MOPED [4], NSGA–II [6] and SPEA2 [25]. The values of parameters of the algorithms are the same as the ones used in [15]. The DTLZ3, DTLZ6 and DTLZ7 problems scalable multi–objective test problems [7] were used. These problems were selected for their scalability, their known and easily measurable Pareto–optimal front, and their multiple suboptimal fronts. Each problem was configured with 3, 6, 9 and 12 objective functions. In all cases the dimension of the decision space was fixed to 15.



(a) DTLZ3



(b) DTLZ6



(c) DTLZ7

Fig. 20.2. Comparative analysis of the computational complexity of the algorithms under study. The first row represents the mean amount of iterations used by each algorithm; the second, the mean intra–iteration CPU operations used for model–building; and, the third, the mean total CPU operations used by the algorithms.

Tests were carried out under the PISA experimental framework [11]. Algorithms' implementations were adapted from the ones provided by their respective authors with the exception of NSGA–II and SPEA2, that were already distributed as part of the framework, and MOPED and MONEDA that were implemented from scratch. For each problem/dimension pair each algorithm was executed 30 times. As assessing the progress of the algorithms in higher dimensions is a complicate matter the MGBM multi–objective optimization cumulative stopping criterion [14] was used.

Table 20.2. Mean CPU running time for each algorithm solving the DTLZ3, DTLZ6 and DTLZ7 problems.

	$M = 3$	$M = 6$	$M = 9$	$M = 12$
DTLZ3				
MONEDA	85.25 s	1.15×10^3 s	3.63×10^3 s	1.49×10^4 s
naïve MIDEA	118.50 s	1.80×10^3 s	5.16×10^3 s	2.51×10^4 s
mrBOA	453.14 s	7.67×10^3 s	2.39×10^4 s	1.25×10^5 s
RM–MEDA	438.08 s	8.08×10^3 s	1.91×10^4 s	1.05×10^5 s
MOPED	142.40 s	1.87×10^3 s	6.28×10^3 s	3.56×10^4 s
NSGA–II	167.20 s	4.74×10^3 s	1.03×10^4 s	5.53×10^4 s
SPEA2	133.56 s	3.98×10^3 s	9.88×10^3 s	4.45×10^4 s
DTLZ6				
MONEDA	107.86 s	1.32×10^3 s	4.14×10^3 s	1.74×10^4 s
naïve MIDEA	139.70 s	2.31×10^3 s	6.15×10^3 s	3.02×10^4 s
mrBOA	554.05 s	9.34×10^3 s	2.84×10^4 s	1.52×10^5 s
RM–MEDA	553.61 s	9.70×10^3 s	2.47×10^4 s	1.19×10^5 s
MOPED	181.88 s	2.29×10^3 s	7.68×10^3 s	4.31×10^4 s
NSGA–II	215.05 s	5.99×10^3 s	1.27×10^4 s	6.86×10^4 s
SPEA2	152.01 s	4.80×10^3 s	1.22×10^4 s	5.27×10^4 s
DTLZ7				
MONEDA	78.19 s	9.62×10^2 s	3.15×10^3 s	1.14×10^4 s
naïve MIDEA	106.20 s	1.60×10^3 s	4.34×10^3 s	2.22×10^4 s
mrBOA	372.48 s	7.20×10^3 s	1.82×10^4 s	1.06×10^5 s
RM–MEDA	368.35 s	7.32×10^3 s	1.52×10^4 s	8.41×10^5 s
MOPED	117.80 s	1.67×10^3 s	5.75×10^3 s	3.07×10^4 s
NSGA–II	151.33 s	3.66×10^3 s	7.76×10^4 s	4.28×10^4 s
SPEA2	119.78 s	3.29×10^3 s	9.11×10^4 s	3.93×10^4 s

Although we are interested on measuring the computational efforts dedicated to the obtaintion of the solutions its is also necessary to assess the quality of the optimizations. After all, we are interested in finding methods capable of successfully tackling multi–objective problems with a viable amount of computation. For this task the hypervolume indicator [11] is used. This indicator measures the volume of the set defined by the solution set and a set of nadir points. Therefore larger values of the indicator represents better solutions.

The indicator values calculated for each algorithm and problem dimension are summarized on table 20.1. There it can be appreciated that MONEDA outperforms the rest of the algorithms in most cases, in particular in higher dimensions.

In order to proceed with the experiments it was measured the amount of floating–point CPU operations used by the optimization itself. This allows the comparison of the different algorithms taking into account only the computation effort dedicated to the optimization without the interference of other aspects of the algorithms or the computational environment in which they are being executed.

Figure 20.2 summarizes the mean number of iterations used by each algorithm, the mean intra–iteration CPU operations used for model–building and the mean total CPU operations used by the algorithms.

The fundamental conclusion that can be extracted from these tests is that MONEDA is the algorithm that better scales in terms of computational complexity. This set of measurements reinforces the conclusions obtained in previous works. Therefore we can draw as primary conclusion that MONEDA is not only an advantageous methods for MOP solving in terms of quality of optimization but also it is capable of doing so with a lesser computational effort when compared with similar approaches. This fact can be attributed to the fast model–building algorithm employed.

Table 20.2 summarizes the mean running time of the different experiments. This data reaffirms the conclusions extracted for far. At the same time, it shows the relatively large duration of the experiments.

It is also very illustrative the case of mrBOA and RM–MEDA that, although they require fewer iterations, its mean CPU operations per iteration is the highest and correspondingly the total amount of CPU operations.

Another interesting phenomenon is the relatively low increase in the number of iterations when moving from 6 to 9 objectives. This attitude is shared across all algorithms. In our opinion it can be attributed to the relatively large size of the population used. It is also noticeable the (presumably) exponential increase on the amount of iterations and the CPU consumption as the problem complexity grows. This is fact consistent with similar studies performed before [10, 21, 5]. This means that future algorithms should be aware of this problem and at least try to alleviate this growth.

20.6 Final Remarks and Future Work

In this paper we performed a comparative study of MONEDA and a set of state–of–the–art MOEDAs and MOEAs. The objective of the experiments was to comprehend the computational impact of MONEDA with regard to similar approaches. A similar study has not been previously proposed. The experiments performed tested the ability of each approach to scalably solve many–objective problems.

The fundamental result obtained here is that MONEDA is not only yields similar or better solutions when compared with other approaches but also does it with at a lower computational cost.

An scheme that reuses computation of previous iterations will probably be usefull in order to further reduce the computational footprint of optimizers, and, perhaps, even improve the quality of the optimization. For our particular case, the model built in an iteration should be reused at a given degree in the subsequent iterations.

Acknowledgements. This work was supported by projects MADRINET, TEC2005-07186-C03-02, SINPROB and TSI2005-07344-C02-02.

References

[1] Ahn, C.W.: Advances in Evolutionary Algorithms. Theory, Design and Practice. Springer, Heidelberg (2006)

[2] Bosman, P.A., Thierens, D.: The Naïve MIDEA: A baseline multi–objective EA. In: Coello Coello, C.A., Hernández Aguirre, A., Zitzler, E. (eds.) EMO 2005. LNCS, vol. 3410, pp. 428–442. Springer, Heidelberg (2005)

[3] Coello Coello, C.A., Lamont, G.B., Van Veldhuizen, D.A.: Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd edn. Genetic and Evolutionary Computation. Springer, New York (2007), `http://www.springer.com/west/home/computer/foundations?SGWI=4-156-22-173660344-0`

[4] Costa, M., Minisci, E., Pasero, E.: An hybrid neural/genetic approach to continuous multi–objective optimization problems. In: Apolloni, B., Marinaro, M., Tagliaferri, R. (eds.) WIRN 2003. LNCS, vol. 2859, pp. 61–69. Springer, Heidelberg (2003)

[5] Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Chichester (2001)

[6] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)

[7] Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable test problems for evolutionary multiobjective optimization. In: Abraham, A., Jain, L., Goldberg, R. (eds.) Evolutionary Multiobjective Optimization: Theoretical Advances and Applications, Advanced Information and Knowledge Processing, pp. 105–145. Springer, Heidelberg (2004)

[8] Ehrgott, M.: Multicriteria Optimization. Lecture Notes in Economics and Mathematical Systems, vol. 491. Springer, Heidelberg (2005)

[9] Fritzke, B.: A growing neural gas network learns topologies. In: Tesauro, G., Touretzky, D.S., Leen, T.K. (eds.) Advances in Neural Information Processing Systems, vol. 7, pp. 625–632. MIT Press, Cambridge (1995)

[10] Khare, V., Yao, X., Deb, K.: Performance Scaling of Multi-objective Evolutionary Algorithms. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 376–390. Springer, Heidelberg (2003)

[11] Knowles, J., Thiele, L., Zitzler, E.: A tutorial on the performance assessment of stochastic multiobjective optimizers. TIK Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich (2006)

[12] Larrañaga, P., Lozano, J.A. (eds.): Estimation of Distribution Algorithms. In: A new tool for Evolutionary Computation. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, Dordrecht (2002)

[13] Levon, J.: OProfile manual. Victoria University of Manchester (2004), `http://oprofile.sourceforge.net/`

[14] Martí, L., García, J., Berlanga, A., Molina, J.M.: A cumulative evidential stopping criterion for multiobjective optimization evolutionary algorithms. In: Thierens, D., Deb, K., Pelikan, M., Beyer, H.G., Doerr, B., Poli, R., Bittari, M. (eds.) Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO 2007, p. 911. ACM Press, New York (2007), `http://portal.acm.org/citation.cfm?doid=1276958.1277141`

[15] Martí, L., García, J., Berlanga, A., Molina, J.M.: Introducing MONEDA: Scalable multiobjective optimization with a neural estimation of distribution algorithm. In: Thierens, D., Deb, K., Pelikan, M., Beyer, H.G., Doerr, B., Poli, R., Bittari, M. (eds.) GECCO 2008: 10th Annual Conference on Genetic and Evolutionary Computation, pp. 689–696. ACM Press, New York (2008); eMO Track "Best Paper" Nominee

[16] Martí, L., García, J., Berlanga, A., Molina, J.M.: Model–building algorithms for multiobjective EDAs: Directions for improvement. In: Michalewicz, Z. (ed.) Computational Intelligence: Research Frontiers, pp. 2848–2855. IEEE Press, Piscataway (2008) doi:10.1109/CEC.2008.4631179

[17] Martí, L., García, J., Berlanga, A., Molina, J.M.: Scalable continuous multiobjective optimization with a neural network–based estimation of distribution algorithm. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, A.Ş., Yang, S. (eds.) EvoWorkshops 2008. LNCS, vol. 4974, pp. 535–544. Springer, Heidelberg (2008)

[18] Martinetz, T.M., Berkovich, S.G., Shulten, K.J.: Neural–gas network for vector quantization and its application to time–series prediction. IEEE Transactions on Neural Networks 4, 558–560 (1993)

[19] Pareto, V.: Cours D'Economie Politique. F. Rouge, Lausanne (1896)

[20] Pelikan, M., Sastry, K., Goldberg, D.E.: Multiobjective estimation of distribution algorithms. In: Pelikan, M., Sastry, K., Cantú-Paz, E. (eds.) Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications. Studies in Computational Intelligence, pp. 223–248. Springer, Heidelberg (2006)

[21] Purshouse, R.C., Fleming, P.J.: On the evolutionary optimization of many conflicting objectives. IEEE Transactions on Evolutionary Computation 11(6), 770–784 (2007)

[22] Qin, A.K., Suganthan, P.N.: Robust growing neural gas algorithm with application in cluster analysis. Neural Networks 17(8–9), 1135–1148 (2004)

[23] Timm, H., Borgelt, C., Doring, C., Kruse, R.: An extension to possibilistic fuzzy cluster analysis. Fuzzy Sets and Systems 147(1), 3–16 (2004)

[24] Zhang, Q., Zhou, A., Jin, Y.: RM–MEDA: A regularity model–based multiobjective estimation of distribution algorithm. IEEE Transactions on Evolutionary Computation 12(1), 41–63 (2008)

[25] Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In: Giannakoglou, K., Tsahalis, D., Periaux, J., Papailou, P., Fogarty, T. (eds.) EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, Athens, Greece, pp. 95–100 (2002)