

# A MULTI-OBJECTIVE EVOLUTIONARY ALGORITHM USING NEURAL NETWORKS TO APPROXIMATE FITNESS EVALUATIONS

A. Gaspar-Cunha<sup>a</sup> and A. Vieira<sup>b</sup>

<sup>a</sup>IPC – Institute for Polymers and Composites, Dept. of Polymer Engineering,  
University of Minho, Campus de Azurém, 4800-058 Guimarães, Portugal  
agc@dep.uminho.pt

<sup>b</sup>Dept. of Physics, Instituto Superior de Engenharia do Porto,  
R. S. Tome, 4200 Porto, Portugal  
asv@isep.ipp.pt

## ABSTRACT

Two different methods to accelerate the search of a Multi-Objective Evolutionary Algorithm (MOEA) using Artificial Neural Networks are presented. Two different methods are proposed. One using ANN to approximate the fitness of the solutions alternated with the real fitness evaluation, being the ANN approximation used only when the estimated error of the neural network was lower than a pre-defined value. In the second method, the ANN is used as a local search strategy by defining new better solutions from the precedent generation. These methods can substantially reduce the number of fitness evaluations on computational expensive problems while not compromise the good search capabilities of MOEA. The efficiency of the methods proposed is tested on several benchmark functions as well on a real multi-optimization problem of polymer extrusion.

**Keywords:** Genetic Algorithms, Neural Networks, Multiobjective Hybrid Algorithm, Engineering Design.

## 1. INTRODUCTION

Multi-Objective Evolutionary Algorithms (MOEAs) are efficient methods to evaluate the Pareto-optimal set in difficult multiobjective optimization problems, such as linear programming and combinatorial optimization. Several MOEA capable of working with a population of points to define an approximation to the Pareto set with a single run have been suggested (Schafer, 1984), (Fonseca et. al., 1993), (Gaspar-Cunha et. al., 1997), (Deb et. al., 2000), (Deb, 2001), (Zitzler et. al., 2001) and (Knowles and Corne, 2000).

One of the major difficulties in applying evolutionary algorithms to real problems is the large number of evaluations of the objective functions necessary to obtain an acceptable solution - typically of the order of several thousands. Often these are time-consuming evaluations obtained by solving numerical codes with expensive methods like finite-differences or finite-elements. Reducing the number of evaluations necessary to reach an acceptable solution is thus of major importance (Nain and Deb, 2002) and (Jin et. al., 2002). This difficulty may be alleviated using distributed computations where each fitness evaluation is performed on a separate processor. However, this requires a large number of networked computers and an adequate parallelization of the numerical code. Finding good approximate methods is even harder for multi-objective problems due to number of criteria and the possible interaction between them.

The use of approximate models in Evolutionary Algorithms (EA) has received little attention, particularly for multiobjective optimization. Several approaches can be used to approximate objective functions, such as statistical methods or Artificial Neural Networks (ANN). Recently Poloni *et al.* (Poloni et. al., 2000) combined several optimization techniques in a

multiobjective optimization problem. They used a multi-objective Genetic Algorithm (GA) to explore the search space, a neural network for data interpolation and a gradient-based technique for local search of the solution. These three techniques are applied sequentially using as initial points the solutions obtained by the genetic algorithm. Since this approach results from the sequential link of the three different optimization techniques, it lacks integration and can not take full advantage of the techniques simultaneously.

Nain and Deb (Nain and Deb, 2002) suggested a different approach to combine GA and ANN. It consists of training an ANN with the set of points obtained by the GA over a certain number of generations. Then, they use the approximate values for the fitness functions, given by the neural network, to evaluate the GA population over a fixed number of generations. The difficulty in this approach is finding the number of generations where the exact and the approximate function evaluation should be used. These numbers are clearly problem dependent and influence decisively the search speed. This approach will be adapted in this paper to work together with our MOEA, and will be identified as Method A1.

Jin *et al.* (Jin *et al.*, 2002) proposed a method similar to the previous authors but implementing an adaptive strategy to select between the use of exact or approximate functions evaluations. The frequencies at which the original function is called and the approximate model is updated are controlled by the determination of the local fidelity of the approximate model. Moreover, they propose a further technique using the covariance matrix to constrain the search algorithm towards promising regions of the search space. However, this hybrid algorithm was only applied to single objective problems.

This work presents a further integration of ANN in MOEA by proposing two different hybrid approaches to estimate the functions or the solutions used by a Multi-Objective Evolutionary Algorithm, namely the Reduced Pareto Set Genetic Algorithm with Elitism (RPSGAe) (Gaspar-Cunha *et al.*, 1997) and (Gaspar-Cunha and Covas, 2004). In the first approach, the algorithm proposed controls dynamically the number of exact and approximate function evaluations through the determination of the error produced by the ANN approximation in each generation. This characteristic eliminates the need of the initial definition of number of generations where the exact and the approximate function evaluations should be used and reduces significantly the algorithm dependency on the ANN parameters. In the second approach, in each generation some tentative solutions are generated by a local search using an ANN. In this case the neural network is used inversely to obtain some solutions resulting from dislocating the approximation to the Pareto front existent in each generation in the direction of the optimization of the various criteria.

This paper is organized as follows. In section 2 the Reduced Pareto Set Genetic Algorithm is briefly described. In section 3 the ANN implementation and characteristics are presented. Section 4 is committed to the detailed description of the two hybrid algorithms proposed. The application of these two methods to some multi-objective benchmark problems is presented in section 5, where it will be demonstrated that the method with error control reduces the need of the initial estimation of the algorithm parameters and the method with inverse ANN is able to produce better solutions. Finally, in section 6 the algorithm will be applied to solve a difficult and computation time consuming multi-objective real world problem on polymer extrusion process.

## 2. MULTIOBJECTIVE OPTIMISATION ALGORITHM

The use of Evolutionary Algorithms as a method to explore and find out an approximation to the Pareto-optimal front on multi-objective optimization problems has been well recognized in the last decade. This is due because the difficulty of the traditional exact methods to solve this type of problems and by its capacity to explore and combine various solutions to find the Pareto front in a single run.

In a multi-objective algorithm the various criteria (or objective) considered are optimized simultaneously. Thus, each individual solution present in the Pareto set explicit a compromise between all the criteria. The solution space can be seen as sets of dominated and non-dominated points. A non-dominated point is a solution at least as good as others with respect to all objectives,

but strictly better with respect to at least one objective. Therefore, one solution point dominates another when it is a good in every objective and formally better in at least one objective (Deb, 2001).

A Multi-Objective Evolutionary Algorithm must provide the homogeneous distribution of the population along the Pareto frontier together with an improvement of the solutions along successive generations. This can be done using three basic operators:

- **Fitness assignment** - to guide the population in the direction of the Pareto frontier using a robust and efficient multiobjective selection method;
- **Density estimation** - to maintain the solutions dispersed trough the entire Pareto frontier using an operator able to take into account the proximity of the solutions;
- **Archiving** - to prevent the deterioration of the fitness during the successive generations by maintaining an external population where the best solutions found so far are kept and are periodically incorporated in the main population.

In this work the Reduced Pareto Set Genetic Algorithm with Elitism (RPSGAe) is adopted (Gaspar-Cunha and Covas, 2004), where a clustering technique is applied to reduce the number of solutions on the efficient frontier. Initially, RPSGAe sorts the population individuals in a number of pre-defined ranks using a clustering technique, in order to reduce the number of solutions on the efficient frontier while maintaining it characteristics intact. Then, the individuals' fitness is calculated through a ranking function. To incorporate this technique, the traditional GA was modified as follows (Gaspar-Cunha and Covas, 2004) and (Gaspar-Cunha, 2000):

#### Algorithm 1:

- 1- Random initial population (internal)
- 2- Empty external population
- 3- **while** not Stop-Condition **do**
  - a) Evaluate internal population
  - b) Calculate the Fitness of the individuals using clustering
  - c) Copy the best individuals to the external population
  - d) **if** the external population becomes full
    - Apply the clustering to this population
    - Copy the best individuals to the internal population
  - end if**
  - e) Select the individuals for reproduction
  - f) Crossover
  - g) Mutation

The algorithm follows the steps of a traditional GA except in the existence of an external (elitist) population and in specific fitness evaluation. Initially, an internal population of size  $N$  is randomly defined and an empty external population formed. At each generation a fixed number of best individuals, obtained by reducing the internal population with the clustering algorithm (Gaspar-Cunha, 2000), are copied to an external population. This process is repeated until the number of individuals of the external population becomes full. Then, the clustering technique is applied to sort the individuals of the external population, and a pre-defined number of the best individuals are incorporated in the internal population by replacing lowest fitness individuals. Detailed information about this algorithm can be found elsewhere (Gaspar-Cunha and Covas, 2004) and (Gaspar-Cunha, 2000).

### 3. NEURAL NETWORKS

Artificial Neural Network (ANN) implemented by a Multilayer Preceptron is a flexible scheme capable of approximating an arbitrary complex function, providing that enough training

data is available (Bishop, 1997). An ANN basically builds a map between a set of inputs and the respective outputs. It is particularly well suited to non-linear regression analysis with noisy signals and incomplete data. A feed-forward neural network consists of an array of input nodes connected to an array of output nodes through successive intermediate layers. Each connection between nodes has a weight, initially random, which is adjusted during a training process. The output of each node of a specific layer is a function of the sum on the weighted signals coming from the previous layer. The crucial points in the construction of an ANN are the selection of inputs and outputs, the architecture of the ANN, that is, the number of layers and of nodes in each layer, and finally, the training algorithm.

### 3.1. Training

In supervised learning, training is performed by presenting a large set of examples, called the training set, to the network. Each example consists of a set of inputs presented to the input layer and the respective set of desired outputs presented to the output layer. Although training an ANN can be time-consuming, once this stage is successfully completed, the input-output mapping is evaluated almost instantaneously. However, care must be taken to use an adequate training set, representative of the sampling space. In many cases this is not feasible, and the sampling space must be restricted to a specific sub-domain. This means that ANNs are best applied to specific, well-defined problems.

Error back-propagation is the simplest and most widely used algorithm to train feed-forward neural networks. In this algorithm the training is performed by minimizing a loss function, usually the sum of square errors over the  $N$  elements of the training set. In this case we used a generalization of the square error function given by:

$$E = \frac{1}{\beta} \sum_{n=1}^N (y_n - \tilde{y}_n)^\beta \quad (1)$$

where  $y_n$  are the true value of the function to be approximated and  $\tilde{y}_n$  the outputs produced by the neural network, and  $\beta$  is a training parameter - for simplicity we consider a single output. The presence of outliers is a common difficulty in some regression problems and is the rule in approximating MOEA objective functions. The error function used here has the advantage that it can control the contribution of outliers to the training by setting  $0 < \beta < 2$ . For  $\beta = 2$ , the error function (1) reduces to the sum of square error.

By minimizing Equation (1) with respect to the weights  $w_{ij}^k$  connecting the node  $i$  of the last hidden layer  $k$  to nodes  $j$  of the output layer, the following recursive relation for updating the weights, after each iteration, is obtained:

$$w_{ij}^k(t+1) = w_{ij}^k(t) + \eta \delta^k s_i + \alpha (w_{ij}^k(t) - w_{ij}^k(t-1)) \quad (2)$$

where  $s_i$  is the output of the node  $i$  in layer  $k$ , and  $\delta^k$ , for the last layer, is given by:

$$\delta^k = y(1-y)(\tilde{y} - y) \quad (3)$$

The weights of layers  $k-1$  are updated by back-propagating the error that corresponds to substituting  $\delta^k$  by:

$$\delta_i^{k-1} = s_i(1-s_i) \sum_j \delta_j^k w_{ij}^{k-1} \quad (4)$$

where  $s_i$  is the output of node  $i$  in layer  $k-1$ . The learning rate parameter  $\eta$  should be adapted to the difficulty of the problem and is usually set below 0.2. The momentum term  $\alpha$  is added to avoid trapping in local minima. The output of each node is taken to be the sigmoid function  $f(s)$  of the weighted sum  $s$  of the outputs from all the nodes of the previous layer:

$$f(s) = (1 + e^{-(s-\theta)})^{-1} \quad (5)$$

where the parameter  $\theta$  is a bias that usually is set to zero. The figure of merit of the network is the final mean square error ( $E_{rms}$ ), given by:

$$E_{rms} = \frac{\sum_{i=1}^{N_t} (y_i - \tilde{y}_i)^2}{\sum_{i=1}^{N_t} (y_i - \bar{y})^2} \quad (6)$$

where  $\bar{y}$  is the average of the target function. For the training set, this error decreases monotonically with the number of training epochs. The training process should not proceed indefinitely to avoid over-fitting to the data and a deterioration of the generalization capabilities of the network. Therefore the training process must be halted by some convergence criterion to guarantee the lowest generalization error.

The usual training criterion is minimization of the mean square error on another sample of examples, not used for training, called the test set. However, in many cases, as in the presented in this work, obtaining training data is computationally expensive, and we have to restrict to small datasets. To avoid over-fitting in these situations early stopping is imposed, which is simply halting the training upon reaching a certain number of training epochs.

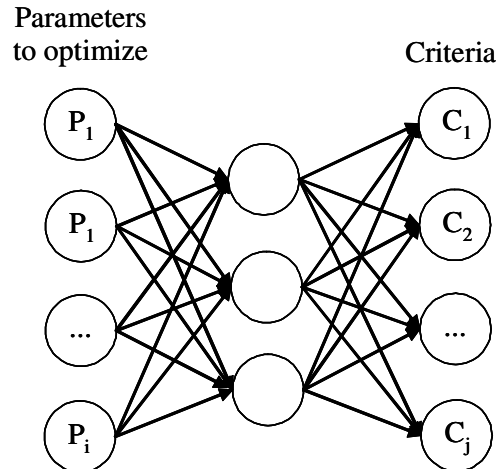
### 3.2. Application of ANN in MOEA

Using ANN with Evolutionary Algorithms is a powerful combination to approach the exploitation/exploration dilemma. By constructing a smooth mapping, ANN can adequately *exploit* specific regions for candidate solutions. ANN are adequate to local search. On the other hand, Evolutionary Algorithms are efficient in *exploring* huge search spaces of multivariate functions with many local minima. They are adequate to global search.

The goal of this work is to implement an approximation to the fitness functions for multi-objective optimization independent of the criteria to be optimized and the parameters of the Evolutionary Algorithm. This is achieved by implementing an automatic algorithm to adjust to the difficulties of training a neural network in an ever changing environment. Let us specify the cause and remedies for these difficulties.

Giving adequate network architecture, training parameters and enough data, an ANN can be trained in order to reach a sufficiently small training error. Thus the ANN is considered to be a good estimator of the data, in the present case the fitness functions for MOEA problems. The performance of the neural network is closely related to the quality of the training data. If training data does not cover all regions of the search space, huge errors may occur due to extrapolation. Errors may also occur when the training points are not sufficiently dense, *i.e.*, insufficient data. Both problems are particularly acute for approximations to functions used in MOEA optimization.

In the present case, the training data is composed of previous exact function evaluations, performed by the evolutionary algorithm. Figure 1 show a representation of the ANN used, where the input variables are the parameters to optimize and the output variables are the criteria evaluated by the fitness function. As the evolutionary algorithm evolves towards the desired solution space, the points available for training will also evolve until they concentrate on a specific region. At each generation new points are obtained covering different regions of the search space. Training the network is therefore a dynamic task that is known as online learning.



**Figure 1.** Scheme of the ANN used.

Online, or dynamic learning, is harder to implement and a careful strategy for constantly retraining the network has to be chosen. If retraining is too slow the algorithm may not adapt sufficiently fast and therefore we end-up with an inadequate network. However, if retraining is too fast, or too much old data is excluded from the training set, the algorithm may end out with a “myopic” network with poor generalization capabilities (Bull, 1999).

It is clear that the adaptation rate as well as the number of points required to train the network, *i.e.*, the size of the training set, are intimately related either to the rate of convergence of the genetic algorithm or the behavior of the functions to be approximated. If the functions have discontinuities or large oscillations, the number of points required to train the network may be very high. This may not be severe provided the search algorithm drives the system fast enough to regions where the functions may have a better behavior. In this situation, an initial high training error can be accepted until a smoother region is reached.

Two approaches may be considered for online training: i) retraining the network as soon a *single* new point is available and ii) retrain the network from scratch with the newest set of exact function evaluations available. The first approach has the advantage of using immediately information as soon as it is available to improve the network. However, it has the drawback that, by retraining only with the most recent points, it introduces a bias in training towards the newest data which may compromise generalization. Although the second approach does not suffer from this difficulty, the size of the training dataset,  $N_r$ , has to be specified. Furthermore, this approach may be more time consuming since the network is retrained from scratch.

#### 4. ALGORITHM PROPOSED

The main objective of the multi-objective hybrid algorithm proposed here is to reduce the number of evaluations of the exact objective functions without losing performance when the Pareto-front must be established. Three different methods will be studied here. Two of them use approximate solutions given by properly trained ANN, that we call method A1 and method A2. The third approach (named Method B) uses the ANN in a reverse way, *i.e.*, the input layer represents the criteria and the output layer the parameters to be optimized (inverse of Figure 1). The aim is to make a local search, starting from some of non-dominated solutions of the previous generation, in order to obtain in each generation some potential better solutions.

##### 4.1. Global Fitness Function Approximation

Method A1 is identical to the approach proposed by Nain and Deb (Nain and Deb, 2002), the difference is related with the MOEA used (in this case the RPSGAE). This approach was implemented, only, for comparison purposes. In both methods the Evolutionary Algorithm needs to run during some  $p$  initial generations to obtain the first set of evaluations necessary to train the

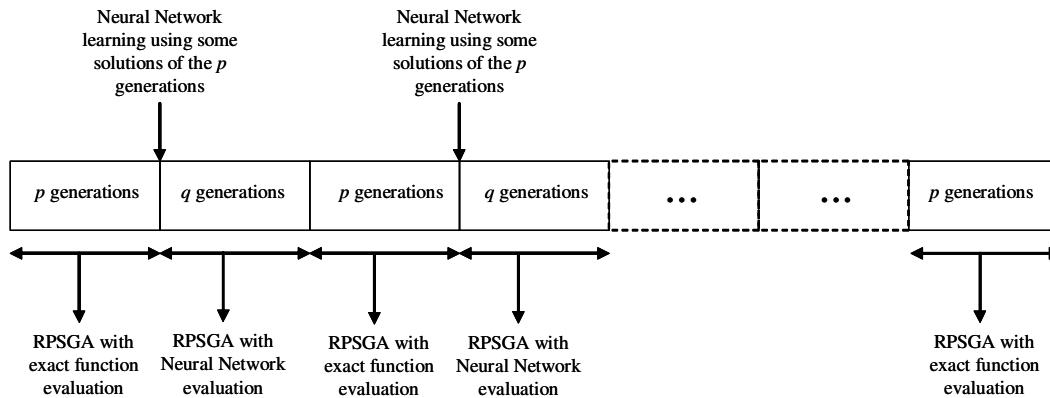
neural network. From this point forward, Method A1 uses the ANN to evaluate all the solutions during a number of consecutive, and pre-fixed,  $q$  generations. Method A2 is similar to A1, but a small portion of individuals,  $M$ , are simultaneously evaluated using the exact function. Thus, all the  $N$  individuals of the population are evaluated using the ANN approximation and  $M$  individuals are evaluated using both the exact evaluation and the ANN.

In many cases training a network to build a sufficiently good approximation to a complex multidimensional function can be very hard. However, we do not need a full approximation over the entire search domain but rather on a specific region where the highest fitness points lies. Thus in order to concentrate on the most promising regions, for both methods we train the ANN only with the best 50% of all available individuals.

Figure 2 illustrates the procedure used in method A1. The fitness of the solutions is determined by exact function evaluation for the first  $p$  generations and using the ANN during the following  $q$  generations. This process is repeated until an acceptable solution is found. The last  $p$  generations are evaluated exactly. The performance of this method depends on the ANN used and on the number of  $p$  and  $q$  generations, *i.e.*, the number of generations in which exact and approximate functions were used.

On the other hand, the accuracy of the ANN is dependent of the number of  $p$  generations that contains the individuals used for training. The large  $p$  the better the ANN approximation is. Conversely, by increasing  $q$  we reduce the number of exact evaluations but at the cost of deteriorating the Pareto-front. Although these two parameters can be pre-defined, their determination is not easy since they are strongly problem dependent. However, some remarks should be made.

Large  $q$  should be avoided, especially for non-smooth functions, as the ANN may give poor approximations based on insufficient training data. It will be expected that a large  $p$  will allow a higher  $q$ . However, this may not be case as training is being performed on an evolving fitness landscape. During the initial generations care must be taken since the network may give large generalization errors, due to the small size of the training set, and thus drive the population to poor regions. Since the convergence of the search algorithm is usually faster in this initial stage it is therefore advisable to start with small  $p$  and  $q$ .



**Figure 2.** Schematic structure of the algorithm identified as method A.

Method A2 has the advantage that both parameters  $p$  and  $q$  are automatically determined using a simple criterion. In this method the error introduced by the approximations ( $e_{NN}$ ) can be directly monitored by:

$$e_{NN} = \frac{\sum_{j=1}^M \sqrt{\sum_{i=1}^R \frac{(C_{i,j}^{NN} - C_{i,j})^2}{R}}}{M} \quad (7)$$

where  $R$  is the number of criteria,  $M$  the number of solutions evaluated using both the exact function and the ANN,  $C_{i,j}^{NN}$  is the value of criteria  $i$  for solution  $j$  evaluated by ANN and  $C_{i,j}$  is the value of criteria  $i$  for solution  $j$  evaluated by exact function.

As the algorithm evolves it may drift to regions outside the domain covered by the initial training points where the approximation from the neural network may not be adequate. The error term allow us to monitor this situation and thus automatically specify the number of  $q$  generations in which the approximated model is used. Thus  $q$  is the number of generations for which the following inequality holds:

$$e_{NN} < e_0 \tag{8}$$

being  $e_0$  a value that can be fixed by the user or adapted over the evolution towards the desired Pareto-front. This value represents the level of error desired and/or admitted by the user between the exact evaluation and the evaluation provided by the ANN.

Figure 3 presents the structure of the algorithm used for method A2. At each generation all  $N$  points are evaluated with the trained ANN while a fraction of  $M$  points are evaluated exactly. The number of generations for which the approximate evaluations are used,  $q$ , is obtained from Equation (8) and the process is repeated until the maximum number of generations is reached.

The performance of Method A2 is weakly dependent on  $p$ . If  $p$  is small the ANN provides a weak approximation, that, nevertheless, is controlled by Equation (8) resulting in a small number  $q$ . On the other hand, an increase in  $p$  results in a consequent increase  $q$ . Thus, the balance between the number of  $p$  and  $q$  generations will be determined by the capacity of the ANN to approximate the evaluation of the solutions, *i.e.*, is not necessary to define *apriori* with greater accuracy the value of  $p$  and  $q$  generations.

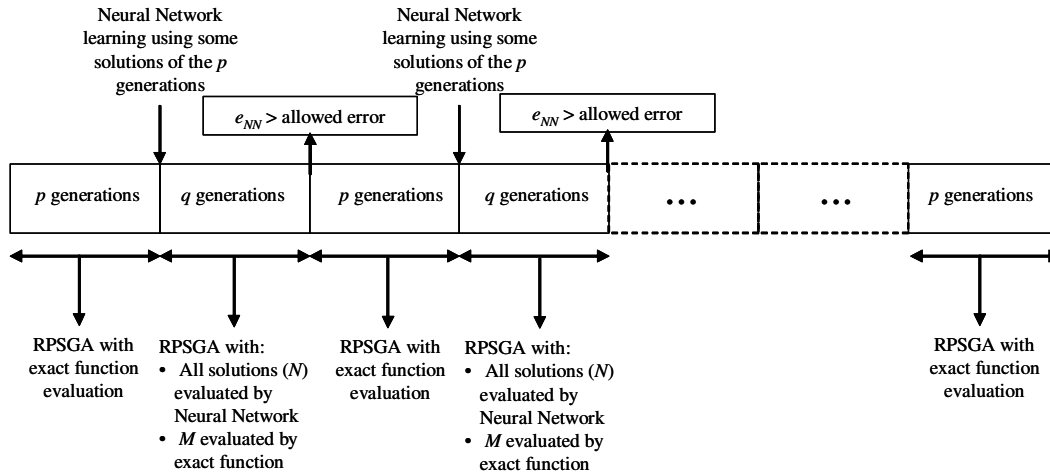
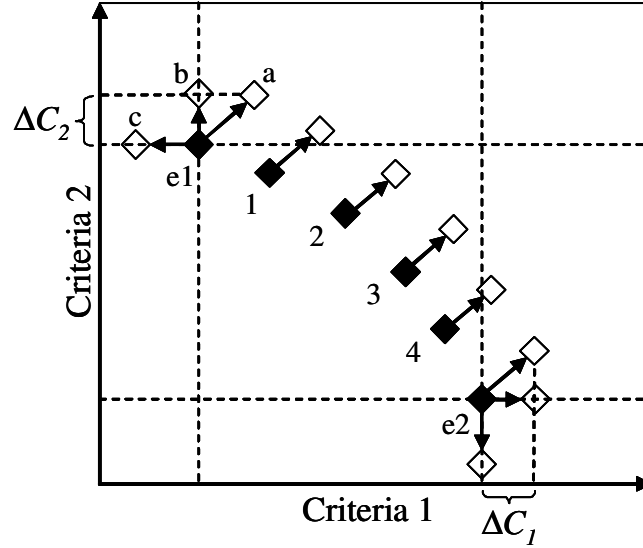


Figure 3. Schematic structure of the algorithm identified as method A2.

#### 4.2. Local Fitness Function Approximation

As stated before, in Method B an Inverse ANN (IANN) is used to carry out a local search, using as starting points some of the non-dominated points of the precedent generation. For that purpose, two additional steps are included in Algorithm 1. One step between 3-b) and 3-c), where the IANN is trained using the values of the present population. The second step after Mutation (step 3-c) where the local search is carried out. Therefore, in this method the individuals for the subsequent generations are obtained by the usual EA operators (crossover and mutation) and by the local search operator (IANN).

Figure 4 illustrates how the local search operator works for a problem with two criteria. This can be easily extended for problems with more criteria. A given percentage of the best individuals of the present generation are selected using the clustering procedure (points 1 to 4 in the example of Figure 4), plus the extreme points (points e1 and e2 in Figure 4). For each extreme point, designated by “e1” (the minimum found for criterion 1) and “e2” (the minimum found for criterion 2), three new individuals are obtained using the IANN. The three new points (identified as a, b and c) are generated with the coordinates calculated as follows (for point e1  $i=1$  and for point e2  $i=2, j = 1, \dots, R$ , being  $R$  the number of criteria):



**Figure 4.** Scheme used for the IANN approach (Method B).

$$\begin{aligned}
 a) \quad & C_j = C'_j + \Delta C_j \\
 b) \quad & C_{j(j=i)} = C'_j \quad \wedge \quad C_{j(j \neq i)} = C'_j + \Delta C_j \\
 c) \quad & C_{j(j=i)} = C'_j - \Delta C_j \quad \wedge \quad C_{j(j \neq i)} = C'_j + \Delta C_j
 \end{aligned} \tag{9}$$

and for the remaining points (points 1 to 4 of Figure 4) the following equation applies:

$$C_j = C'_j + \Delta C_j \quad (j = 1, \dots, R) \tag{10}$$

where  $\Delta C_j$  is the displacement applied to each criteria value, as illustrated in Figure 4. The value chosen for  $\Delta C_j$  must be carefully defined. In this work, since the criteria values are normalized in the interval  $[0,1]$ , the optimal values found are located in the range  $[0.3; 0.4]$ .

## 5. APPLICATION TO BENCHMARK PROBLEMS

The proposed method was tested using the ZDT1, ZDT2, ZDT3, ZDT4 and ZDT6 bi-objective functions – see Table 1. These functions cover various types of Pareto-optimal fronts, such as convex (ZDT1), non-convex (ZDT2), discrete (ZDT3), multimodal (ZDT4) and non-uniform (ZDT6) (Zitzler et. al., 2000).

**Table 1.** Test functions.

Name	m	Equation	$x_i$	Equation
ZDT1	30	$f_1(x_1) = x_1$ $f_2(x_2, \dots, x_m) = g \times \left( 1 - \sqrt{f_1/g} \right)$ where, $g(x_2, \dots, x_m) = 1 + 9 \frac{\sum_{i=2}^m x_i}{m-1}$	$x_i \in [0,1]$	(11)
ZDT2	30	$f_1(x_1) = x_1$ $f_2(x_2, \dots, x_m) = g \times \left( 1 - \left( \frac{f_1}{g} \right)^2 \right)$ where, $g(x_2, \dots, x_m) = 1 + 9 \frac{\sum_{i=2}^m x_i}{m-1}$	$x_i \in [0,1]$	(12)
ZDT3	30	$f_1(x_1) = x_1$ $f_2(x_2, \dots, x_m) = g \times \left( 1 - \sqrt{f_1/g} - \left( \frac{f_1}{g} \right) \sin(10\pi f_1) \right)$ where, $g(x_2, \dots, x_m) = 1 + 9 \frac{\sum_{i=2}^m x_i}{m-1}$	$x_i \in [0,1]$	(13)
ZDT4	10	$f_1(x_1) = x_1$ $f_2(x_2, \dots, x_m) = g \times \left( 1 - \sqrt{f_1/g} \right)$ where, $g(x_2, \dots, x_m) = 1 + 10(m-1) + \sum_{i=2}^m (x_i^2 - 10 \cos(4\pi x_i))$	$x_1 \in [0,1]$ $x_i \in [-5,5]$	(14)
ZDT6	10	$f_1(x_1) = 1 - \exp(-4x_1) \sin^6(6\pi x_1)$ $f_2(x_2, \dots, x_m) = g \times \left( 1 - \left( \frac{f_1}{g} \right)^2 \right)$ where, $g(x_2, \dots, x_m) = 1 + 9 \left( \frac{\sum_{i=2}^m x_i}{m-1} \right)^{0.25}$	$x_i \in [0,1]$	(15)

**5.1. Neural Network approximations**

Since certain functions may have wide variations, the following transformation was applied:

$$y' = \ln(y + b) \tag{16}$$

where the threshold  $b$  was set close to unity to avoid extreme importance to small values. This transformation is necessary to prevent overflow of the weights during training resulting from the presence of large functions values. The outputs were then normalized between 0 and 1.

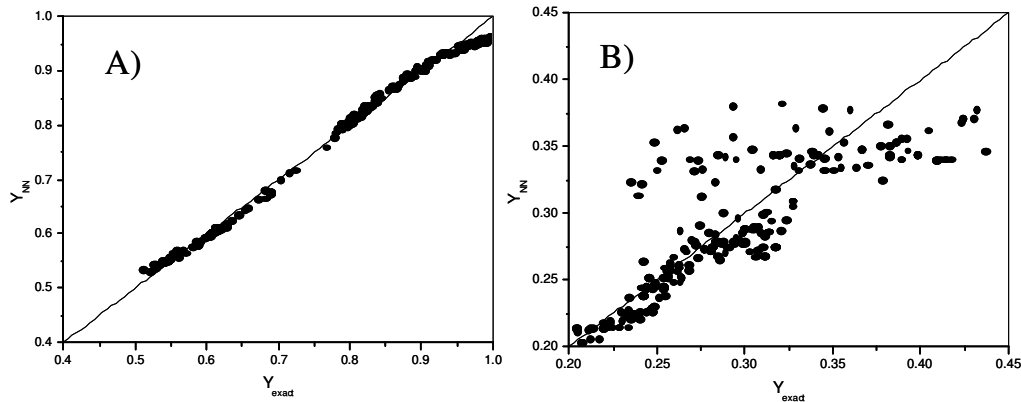
For each training set the inputs were normalized by subtracting the mean and dividing by the standard deviation:

$$x'_i = \frac{x_i - \bar{x}}{\sigma_x} \tag{17}$$

Thus, most values will lie in the range between -1 and 1. We suppose that the search domain is known as well as the maximum and minimum of the functions.

Figure 5 compares the approximation obtained by ANN with the exact values for the ZDT3 function, one of the hardest. The training data used was composed of 750 random solutions correspondent to valid individuals obtained by the EA. The neural network used has 10 neurons on a single hidden layer a learning rate of 0.1 and a momentum of 0.25. In this case, training the network took about 20 s to be completed in a Pentium IV computer running at 1 GHz.

The agreement for function  $f_1$  is excellent while for function  $f_2$  we notice that the neural network overestimates the exact values in the range between 0.25 and 0.30 and underestimates exact function in the range between 0.40 and 0.45. Apart of these small discrepancies we may consider the neural network a good estimator of both functions and the mean square training error obtained was just 1.9%. Note that this data corresponds to the first points obtained by the evolutionary algorithm.



**Figure 5.** Comparison of exact functions ( $Y_{exact}$ ) with approximations given by the neural network ( $Y_{NN}$ ) for the two functions of ZDT3: **a)**  $f_1$  and **b)**  $f_2$ .

## 5.2. Performance metrics

The performance of this algorithm should be measured by a balance between the quality of the solutions obtained and the number of exact objective function evaluations performed. Since the computation time required to train and test the neural network is negligible, when computational demanding problems are to be solved, the number of *exact* evaluations of the objective functions was used as the significant running parameter.

In order to compare the results obtained an evaluation of the performance along the successive generations is needed. However, measuring the quality of solutions in multiobjective optimization is not straightforward due to the presence of multiple, often conflicting, goals. These goals can be: archiving a good approximation to the Pareto-front, having a uniform distribution of the solutions and maximizing the coverage of the Pareto-front. The S-metric proposed by Zitzler (Zitzler, 1999) will be adopted, since it is adequate for problems with few objective dimensions (Knowles and Corne, 2002) and allows the quantification of performance along the successive generations. For this purpose each run was performed 5 times in order to take into account the variation of the random initial population. The RPSGAe uses a population size of 100, 300 generations, a roulette wheel selection strategy, a crossover probability of 0.8, a mutation probability of 0.05, a number of ranks of 30 and the limits of indifference of the clustering technique of 0.01 (Gaspar-Cunha and Covas, 2004).

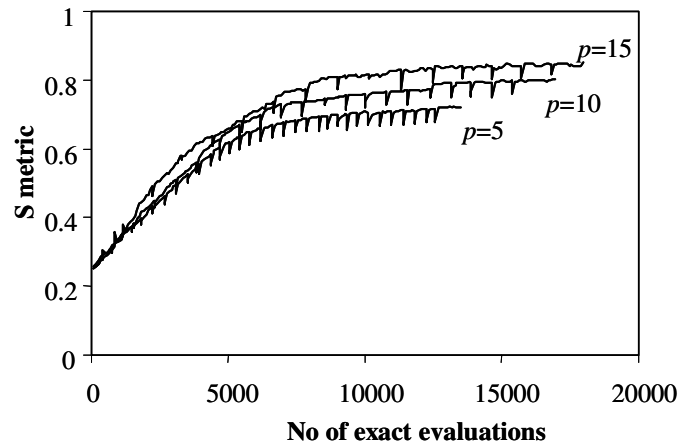
## 5.3. Results from method A1

Initially, the relevance of some parameters on the algorithm performance are studied, namely: number of generations evaluated by the exact function, ( $p = 5, 10$  and  $15$  generations), number of generations evaluated by the approximate model, ( $q = 5, 10$  and  $15$  generations),

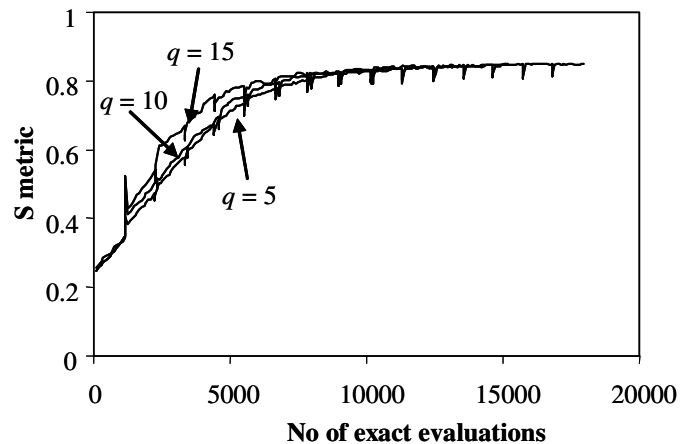
number of hidden neurons of the neural network, ( $N_h = 10, 20$  and  $30$  neurons), learning rate of the neural network, ( $\eta = 0.1, 0.2, 0.3$  and  $0.4$ ) and momentum term, ( $\alpha = 0.2, 0.4, 0.6$  and  $0.8$ ).

For each generation the algorithm calculates the average of the S-metric over the 5 runs performed for each case as a function of the number of *exact* evaluations effectuated so far. The best results obtained for each test problem were compared with the ones obtained using RPSGAe alone. Figures 6 to 9 shows the influence of the number of  $p$  and  $q$  generations, the number of hidden neurons and the learning rate on the algorithm performance for the ZDT1 test problem.

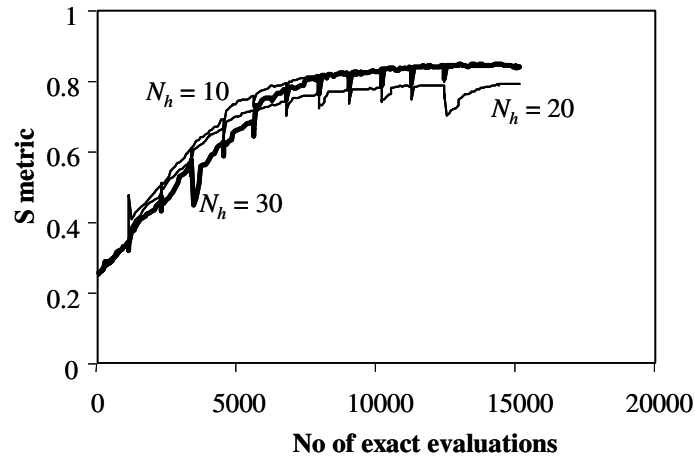
From Figures 6 and 7 is possible to see that using small  $p$  or  $q$  leads to no noticeable benefit since either the training data is insufficient to train the network or the ANN is applied over only a small number of generations. It is known that an ANN with a large number of hidden nodes  $N_h$  is capable to approximate more complex functions. However, Figure 8 shows that this is not always the case. During the initial generations, the worst performance was achieved by the largest neural network containing  $N_h = 30$ . This can be explained by the fact that a larger network needs more training points to be properly constrained, and therefore the number  $N_h$  should follow  $p$ . To avoid the risk of becoming trapped on local minima, it is not advisable to use high learning rates, as becomes clear in Figure 9. From these figures the best set of parameters for the ZDT1 function found were:  $p = q = 15, N_h = 10, \eta = \alpha = 0.2$ .



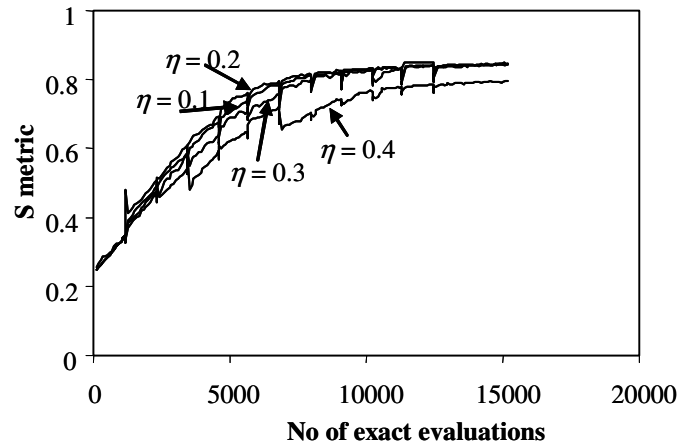
**Figure 6.** Influence of the number of  $p$  generations on the algorithm performance.



**Figure 7.** Influence of the number of  $q$  generations on the algorithm performance.



**Figure 8.** Influence of the number of hidden neurons on the algorithm performance



**Figure 9.** Influence of the learning rate on the algorithm performance

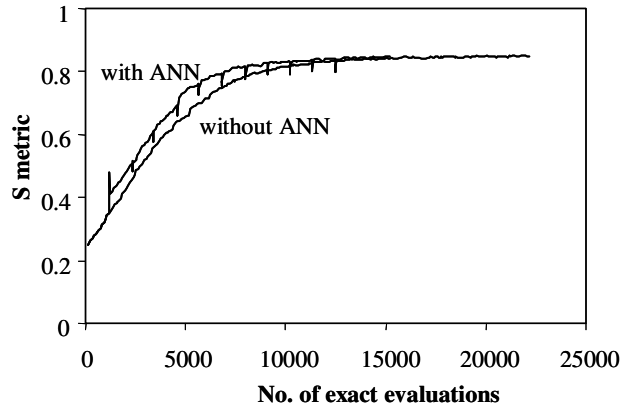
Table 2 presents a summary of this analysis for all the test problems studied. Notice the strong dependence of associated algorithm parameters for method A1. Problems ZDT1 and ZDT6 should be the easiest for the ANN since  $q$  is large and  $N_h$  small. Conversely ZDT3 should be the problem that poses more difficulties to the ANN.

**Table 2.** Best algorithm parameters.

Test problem	$p$	$q$	$N_h$	$\eta$	$\alpha$
ZDT1	15	15	10	0.2	0.2
ZDT2	15	15	30	0.3	0.6
ZDT3	15	5	20	0.3	0.2
ZDT4	15	5	10	0.2	0.6
ZDT6	15	15	10	0.2	0.3

Figure 10 compares the results obtained with traditional RPSGAe and the best results obtained with method A1. Note that, for a given number of exact evaluations, the S-metric is always superior when the approximate method is used. This is particularly visible in the early

convergence where the improvements are larger. After about 10 000 exact evaluations both curves merge to a single plateau where the solution is asymptotically approached. It is possible to conclude that this method saves about 35% of exact function evaluations.



**Figure 10.** Comparison of S-metric with and without ANN for the ZDT1 function (The parameters identified in table 2 were used.)

#### 5.4. Results obtained with method A2

In order to achieve a clear comparison of the performance of our method the following criterion is used:

$$S^* = \frac{\bar{S}_{NN} - \bar{S}}{\bar{S}_{NN}} \quad (18)$$

where,  $\bar{S}_{NN}$  and  $\bar{S}$  are the averages of the S-metric obtained with and without ANN, respectively.

In Figure 11 both  $S^*$  and the difference of exact evaluations as a function of the number of generations are plotted. The difference of exact evaluations can be obtained using Equation (18) by replacing the averages of the S-metric with the averages of the number of exact evaluations for the same runs. When Method A1 is used for small differences of the S-metric, less than 1%, there is an improvement of about 35% in the number of evaluations (Figure 11). These results are obtained after 17 optimization runs where all the algorithm parameters are assessed.

To avoid the cumbersome assessment of the algorithm parameters, required by method A1, method A2 was applied instead to the same example. Various runs using different error levels  $e_0$  (see Equation 8) were performed:  $e_0 = 3, 4, 5$  and 10%. Figure 11 presents the results obtained using an allowed error of 3%. An upper limit for  $p$  and  $q$  was established,  $p_{\max} = 20$  and  $q_{\max} = 30$ , while setting  $N_h = 10$ ,  $\eta = 0.2$  and  $\alpha = 0.25$ .

The number of exact evaluations to reach the same S-metric is also reduced to about 28%, as for method A1 (see Figure 11). However, method A2 has the advantage that no parameter optimization is needed and therefore results are obtained in a single run. Similar results were achieved with different levels of allowed errors, resulting in a decrease of the number of exact evaluations necessary as the error increases.

The same type of analysis was done for the remaining test problems, *i.e.*, 17 runs for method A1, in order to set the algorithm parameters, and 4 runs for method A2 corresponding to different allowed errors of 3, 4, 5 and 10% - see figures 12 to 15. Since ZDT2, Figure 12, is a non-convex problem, it was more difficult to solve. For a level of error of 3% the improvement obtained in the number of evaluations for both schemes were approximately 25%. Identical results were obtained for the ZDT3 problem, Figure 13, an improvement of 23% in the number of evaluations was achieved for an error of  $e_0 = 3\%$ .

The multimodal characteristics of the ZDT4 problem, Figure 14, present an additional difficulty due to the fact that different initial populations, *i.e.*, different seed values, produce different Pareto fronts. Despite this difficulty a reduction of 25% and 10% in the number of exact

evaluations were achieved respectively for method A1 and A2. Finally, Figure 15 presents the results obtained for the ZDT6 non-uniform problem. As expected, the approximation produced by the ANN is weak. Consequently, only a reduction of 9% in the number of evaluations is obtained.

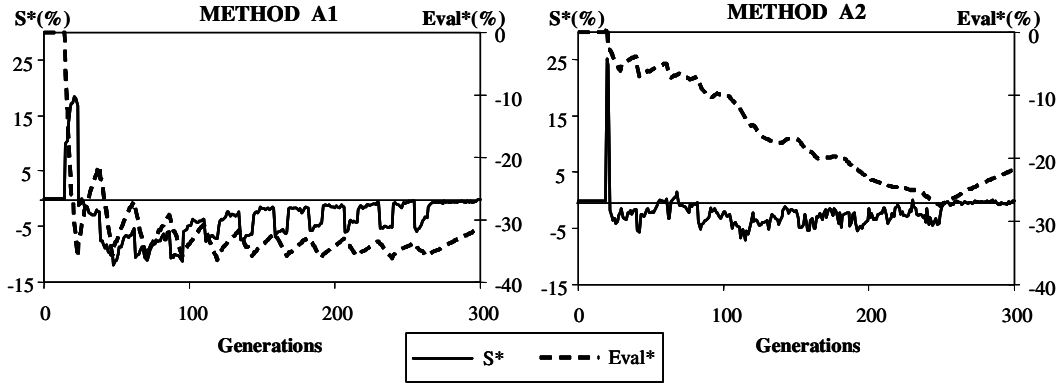


Figure 11. Evolution of the S metric and number of evaluations differences for ZDT1 test problem, using methods A1 and A2

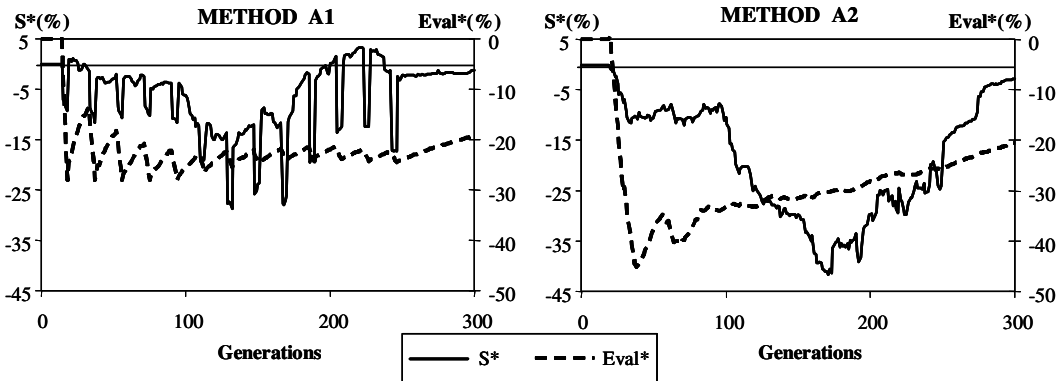


Figure 12. Evolution of the S metric and number of evaluations differences for ZDT2 test problem, using methods A1 and A2

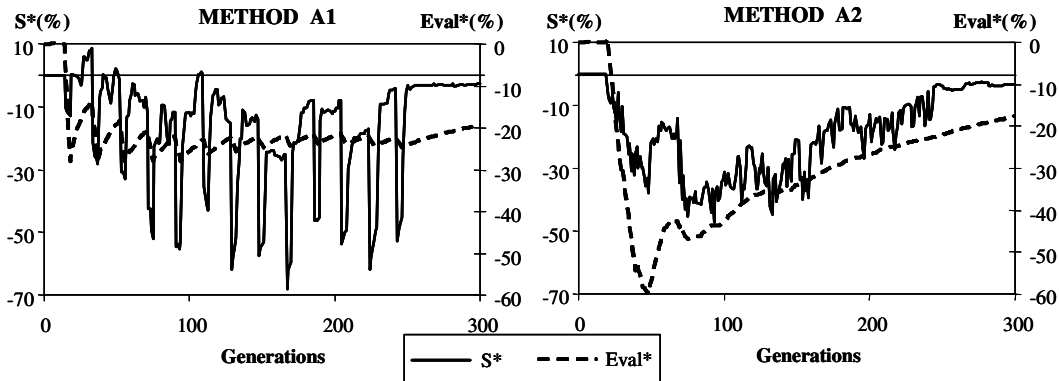


Figure 13. Evolution of the S metric and number of evaluations differences for ZDT3 test problem, using methods A1 and A2

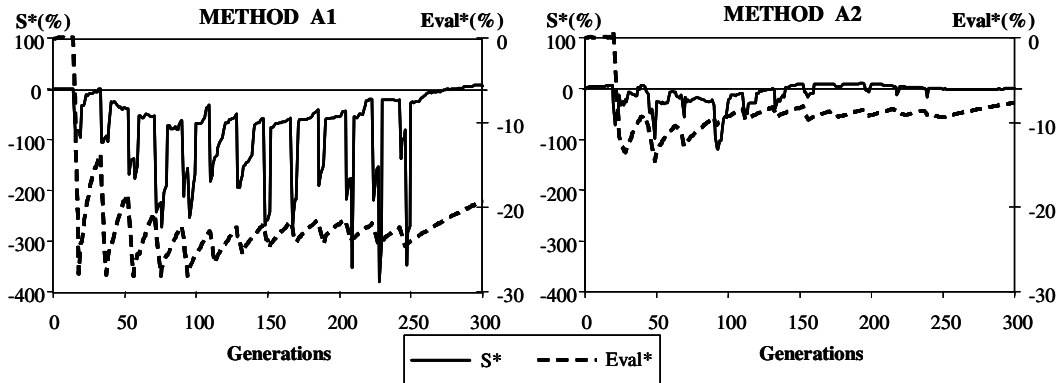


Figure 14. Evolution of the S metric and number of evaluations differences for ZDT4 test problem, using methods A1 and A2

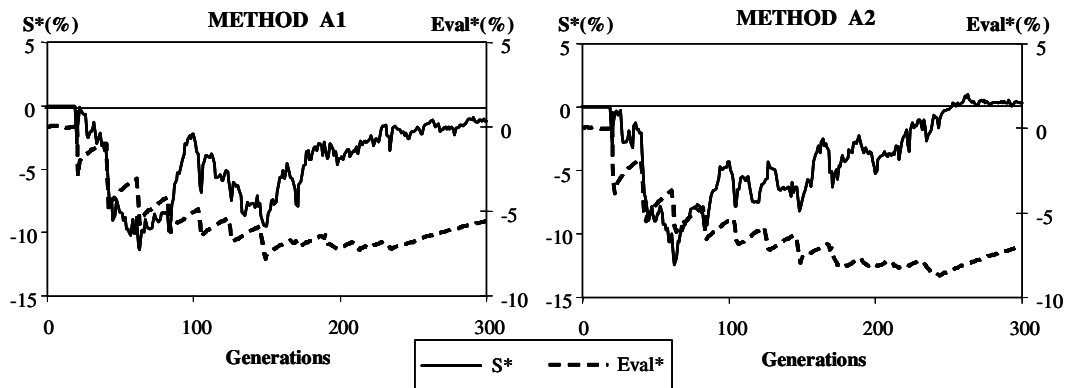


Figure 15. Evolution of the S metric and number of evaluations differences for ZDT6 test problem, using methods A1 and A2

**5.6. Results obtained with method B**

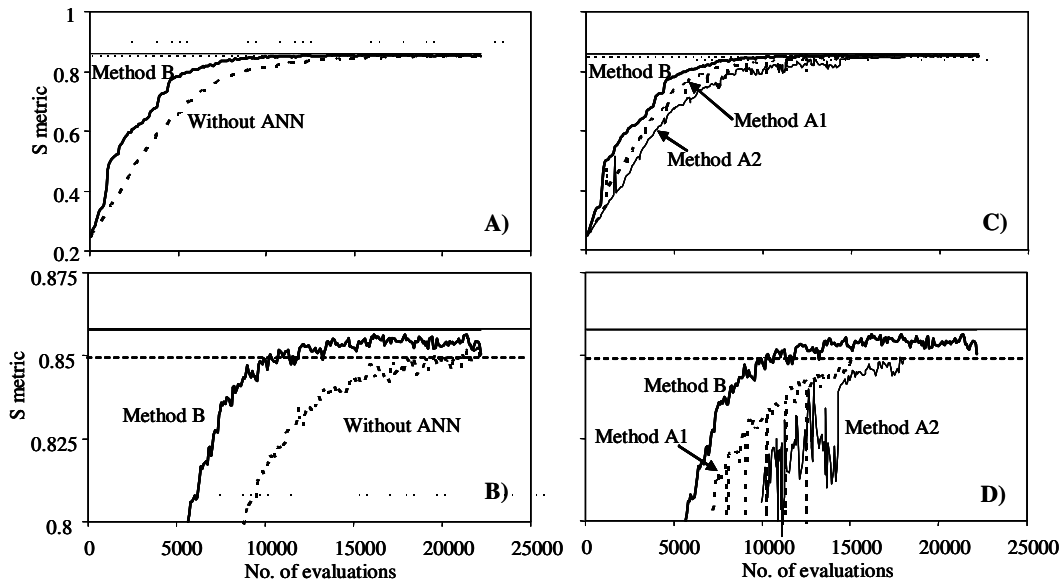
Figure 16 presents a comparison between the three methods and the results obtained without the use of the ANN approximation, *i.e.*, the RPSGAe. The values were obtained by averaging over 5 runs using different initial populations. The horizontal full line represents the S metric of an optimal population, *i.e.*, the optimal S metric, and the horizontal dotted line is located 1% below the first line. Method B is able to reach 99% percent of the optimal S-metric after only 10000 exact function evaluations, which is about half of the 19000 evaluations required by the RPSGAe.

Table 3 summarizes the comparison study carried out between the results obtained using Method B and the RPSGAe alone. The comparison is made using the S metric value after 22000 exact evaluations of the objective function and the number of evaluations to attain the maximum value of the S metric accomplished by the RPSGAe.

Method B always outperforms the RPSGAe in producing a better and faster approximation to the optimal Pareto front.

**Table 3.** Comparison between Method B and RPSGAe without ANN approximation.

Test problem	S metric, 22000 evaluations			Number of evaluations		
	Method B	RPSGAe	Increase (%)	Method B	RPSGAe	Increase (%)
ZDT1	0.851	0.849	<b>0.24</b>	10000	19000	<b>47.4</b>
ZDT2	0.786	0.773	<b>1.68</b>	15300	22000	<b>30.5</b>
ZDT3	2.736	2.554	<b>7.13</b>	18000	22000	<b>18.2</b>
ZDT4	0.1116	0.0807	<b>38.29</b>	5000	22000	<b>77.3</b>
ZDT6	0.599	0.571	<b>4.90</b>	12500	22000	<b>43.2</b>

**Figure 16.** Comparison of the S metric as a function of the number of evaluations for ZDT1 problem using Method B

## 6. APPLICATION TO A REAL OPTIMISATION PROBLEM

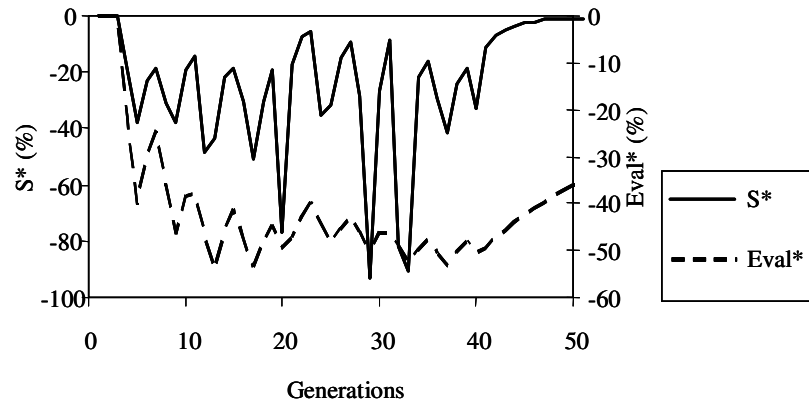
In this section the approximate method A2 will be applied to the screw geometry optimization of a single-screw polymer extruder. Extrusion is a polymer processing technology largely used in the plastics industry in the last 40 years to produce films, sheets, pipes, profiles, fibers, filaments, bottles and containers.

The extruder is characterized by an Archimedes-type screw that rotates inside a heated barrel. The extruder receives the solid pellets at the inlet and melts, mix and homogenize the material. Then, the melted polymer is pumped through the die in order to produce an extrudate with a prescribed cross-section. For modeling purposes, the process is considered as a succession of functional zones characterized by stress, mass, heat or force balances, coupled by adequate boundary conditions in the interface between the zones. The resolution of these differential equations is performed through the method of finite differences. A detailed description of these models and the required optimization can be found elsewhere (Gaspar-Cunha and Covas, 2004) and (Gaspar-Cunha and Covas, 2001). Recently the process was proposed as a real test problem for EMO algorithms and was made available through the internet to the EMO community (Gaspar-Cunha and Covas, 2003) and (Gaspar-Cunha, 2003).

Method A2 is applied in this case in order to reduce the number of exact evaluations of a MOEA applied to the problem of determining the geometry of a conventional screw extruded that

simultaneously maximize the mass output and the mixing degree. The screw geometry is parameterized by the screw lengths of zone 1 and 2,  $L_1 \in [100, 400]$  mm and  $L_2 \in [170, 400]$  mm and by the screw internal diameter of zones 1 and 3,  $D_1 \in [20, 26]$  mm and  $D_3 \in [26, 32]$  mm, the total length of the screw is maintained constant [19].

Ten runs are carried out, five using the RPSGAe without neural networks and five using method B with an allowed error of 3%. The upper limit of  $p$  and  $q$  was established as 3 and 10, respectively. The ANN parameters are settled to  $N_h = 10$ ,  $\eta = 0.2$  and  $\alpha = 0.25$ . Figure 14 shows the evolution of  $S^*$  and the difference of exact evaluations as a function of the number of generations. The improvement obtained in reducing the number of exact function evaluations necessary is approximately 40%. This implies a reduction from 8.5 to 6.0 hours on the computation time when a PC with an AMD processor at 1666 MHz is used.



**Figure 17.** Evolution of the S metric and number of evaluations differences for polymer extrusion problem, using method A2

## 7. CONCLUSIONS

In this work it is shown that Artificial Neuronal Networks are efficient to approximate objective function for multi-objective optimization with evolutionary algorithms. This approach can substantially reduce the large number of function evaluations required by evolutionary algorithms to reach an acceptable Pareto-front.

The efficiency of the approach proposed strongly depends on the difficulty of the functions to be optimized and on the desired degree of approximation. The right approximation degree has to be adjusted: a conservative approximation has no relevant reduction in computation time while an aggressive approach lead to large errors in objective functions and poor Pareto-fronts. Two methods to select an adequate approximation degree have been proposed: one based on an automatic adjustment of the parameters that control the training of the ANN and the generalization error, and other using a local search for better individuals based on an inverse ANN.

Both methods were applied to several benchmark problems and to a real world problem of polymer extrusion. These approaches save considerable computational time, ranging from 13% to about 50%. These savings are particularly relevant when the evaluation of the solutions involves the use of numerical methods with large computational costs, such the real optimization problem on polymer extrusion tested here. For this problem a reduction of more than 6 hours of computational time was achieved for a single optimization run.

## ACKNOWLEDGEMENTS

This work was supported by the Portuguese Fundação para a Ciência e Tecnologia under grant POCTI/EME/48448/2002.

## REFERENCES

- Bishop, C., (1997) *Neural Networks for Pattern Recognition*, Oxford University Press.
- Bull, L., (1999) On Model-based Evolutionary Computation, *Soft. Comp*, **3**, 76.
- Deb, K., Agrawal, S., Pratap, A. and Meyarivan, T. (2000) A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGAI, *Proceedings of the Parallel Problem Solving from Nature VI (PPSNVI)*, 849-858.
- Deb, K. (2001) *Multi-Objective Optimization using Evolutionary Algorithms*, Wiley.
- Fonseca, C.M. and Fleming, P.J., (1993) Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization, *Proc. Fifth Int. Conf. on Genetic Algorithms*, Morgan Kaufman, 416-423.
- Gaspar-Cunha, A., Oliveira, P. and Covas, J.A., (1997) Use of Genetic Algorithms in Multicriteria Optimization to Solve Industrial Problems, *Seventh Int. Conf. on Genetic Algorithms*, Michigan, USA.
- Gaspar-Cunha, A. (2000) *Modelling and Optimization of Single Screw Extrusion*, PhD Thesis, University of Minho, Guimarães, Portugal, (Downloadable from website <http://www.lania.mx/~ccoello/EMOO/>).
- Gaspar-Cunha, A. and Covas, J.A., (2001) The Design of Extrusion Screws: An Optimisation Approach, *International Polymer Processing*, **16** (4), 229-240.
- Gaspar-Cunha, A. and Covas, J.A., (2003) A Real-World Test Problem for EMO Algorithms, *Proceedings of the EMO'2003 conference*, Faro, Portugal.
- Gaspar-Cunha, A. (2003) A Real-World Test Problem for EMO Algorithms, WEB site: <http://www.dep.uminho.pt/pp/index.php3?gaspar@dep.uminho.pt>.
- Gaspar-Cunha, A. and Covas, J.A., (2004) - RPSGAe - A Multiobjective Genetic Algorithm with Elitism: Application to Polymer Extrusion, in a Lecture Notes in Economics and Mathematical Systems volume, Springer.
- Jin, Y., Olhofer, M. and Sendhof, B. (2002) A Framework for Evolutionary Optimization with Approximate Fitness Functions, *IEEE Trans. on Evolutionary Computations* **6** (4), 481-494.
- Knowles, J.D. and Corne, D.W., (2000) Approximating the Non-dominated Front using the Pareto Archived Evolutionary Strategy, *Evolutionary Computation Journal*, **8** (2), 149-172.
- Knowles, J.D. and Corne, D.W., (2002) On Metrics for Comparing Non-Dominated Sets. In *Proceedings of the 2002 Congress on Evolutionary Computation Conference (CEC02)*, IEEE Press, 711-716.
- Nain, P.K.S. and Deb, K., (2002) A Computationally Effective Multi-Objective Search and Optimization Technique Using Coarse-to-Fine Grain Modelling, Kangal Report No. 2002005, (Downloadable from website <http://www.iitk.ac.in/kangal/deb.htm>).
- Poloni, C., Giurgevich, A., Onesti, L. and Pedirola, V., (2000) Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for complex design problem in fluid dynamics, *Computer Methods in Applied Mechanics and Engineering* **186**, 403-420.
- Schafer, J.D., (1984) *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*, Ph. D. Thesis, Nashville, TN, Vanderbilt University.
- Zitzler, E., (1999) *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, PhD Thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.
- Zitzler, E., Deb, K. and Thiele, L., (2000) Comparison of Multiobjective Evolutionary Algorithms: Empirical Results, *Evolutionary Computation*, **8** (2), 173-195.
- Zitzler, E., Laumanns, M. and Thiele, L., (2001) SPEA2: Improving the Strength Pareto Evolutionary Algorithm, TIK report no. 103, Swiss Federal Institute of Technology, Zürich, Switzerland, (Downloadable from website <http://www.tik.ee.ethz.ch/~zitzler/>).