A Study of the Parallelization of a Coevolutionary Multi-Objective Evolutionary Algorithm

Carlos A. Coello Coello and Margarita Reyes Sierra

CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Ingeniería Eléctrica, Sección de Computación
Av. IPN No. 2508. Col. San Pedro Zacatenco, México D.F. 07300, MÉXICO
ccoello@cs.cinvestav.mx, mreyes@computacion.cs.cinvestav.mx

Abstract. In this paper, we present a parallel version of a multi-objective evolutionary algorithm that incorporates some coevolutionary concepts. Such an algorithm was previosly developed by the authors. Two approaches were adopted to parallelize our algorithm (both of them based on a master-slave scheme): one uses Pthreads (shared memory) and the other one uses MPI (distributed memory). We conduct a small comparative study to analyze the impact that the parallelization has on performance. Our results indicate that both parallel versions produce important improvements in the execution times of the algorithm (with respect to the serial version) while keeping the quality of the results obtained.

1 Introduction

The use of coevolutionary mechanisms has been scarce in the evolutionary multiobjective optimization literature [1]. Coevolution has strong links with game theory and its suitability for the generation of "trade-offs" (which is the basis for multiobjective optimization) is, therefore, rather obvious. This paper extends our proposal for a coevolutionary multi-objective optimization approach presented in [2]. The main idea of our coevolutionary multi-objective algorithm is to obtain information along the evolutionary process as to subdivide the search space into n subregions, and then to use a subpopulation for each of these subregions. At each generation, these different subpopulations (which evolve independently using Fonseca & Fleming's ranking scheme [3]) "cooperate" and "compete" among themselves and from these different processes we obtain a single Pareto front. The size of each subpopulation is adjusted based on their contribution to the current Pareto front (i.e., subpopulations which contributed more are allowed a larger population size and viceversa). The approach uses the adaptive grid proposed in [4] to store the nondominated vectors obtained along the evolutionary process, enforcing a more uniform distribution of such vectors along the Pareto front.

This paper presents the first attempt to parallelize a coevolutionary multi-objective optimization algorithm. The main motivation for such parallelization is because the proposed algorithm is intended for real-world applications (mainly in engineering) and therefore, the availability of a more efficient version of the algorithm (in terms of CPU time required) is desirable. In this paper, we compare the serial version of our algorithm (as reported in [2]) with respect to two parallel versions (one that uses Pthreads and

another one that uses MPI). A comparison with respect to PAES [4] is also included to give a general idea of the performance of the serial version of our algorithm with respect to other approaches. However, for a more detailed comparative study the reader should refer to [2]. The main aim of this study is to compare the performance gains obtained by the parallelization of the algorithm. Such performance is measured both in terms of the computational times required as well as in terms of the quality of the results obtained.

2 Statement of the Problem

We are interested in solving problems of the type:

$$\text{minimize } [f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x})] \tag{1}$$

subject to:

$$g_i(\boldsymbol{x}) \geq 0 \quad i = 1, 2, \ldots, m \tag{2}$$

$$h_i(\boldsymbol{x}) = 0 \quad i = 1, 2, \ldots, p \tag{3}$$

where k is the number of objective functions $f_i : R^n \rightarrow R$. We call $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^T$ the vector of decision variables. We thus wish to determine from the set $\mathcal{F}$ of all the vectors that satisfy (2) and (3) to the vector $x_1^*, x_2^*, \ldots, x_n^*$ that are *Pareto optimal*. We say that a vector of decision variables $\boldsymbol{x}^* \in \mathcal{F}$ is *Pareto optimum* if there does not exist another $\boldsymbol{x} \in \mathcal{F}$ such that $f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{x}^*)$ for every $i = 1, \ldots, k$ and $f_j(\boldsymbol{x}) < f_j(\boldsymbol{x}^*)$ for at least one j. The vectors $\boldsymbol{x}^*$ corresponding to the solutions included in the Pareto optimal set are called *nondominated*. The objective function values corresponding to the elements of the Pareto optimal set are called the *Pareto front* of the problem.

3 Coevolution

Coevolution refers to a reciprocal evolutionary change between species that interact with each other. The relationships between the populations of two different species can be described considering all their possible types of interactions. Such interaction can be positive or negative depending on the consequences that such interaction produces on the population. Evolutionary computation researchers have developed several coevolutionary approaches in which normally two or more species relate to each other using any of the possible relationships, mainly competitive (e.g., [5]) or cooperative (e.g., [6]) relationships. Also, in most cases, such species evolve independently through a genetic algorithm. The key issue in these coevolutionary algorithms is that the fitness of an individual in a population depends on the individuals of a different population.

4 Description of the Serial Version of our Algorithm

The main idea of our approach is to try to focus the search efforts only towards the promising regions of the search space. In order to determine what regions of the search space are promising, our algorithm performs a relatively simple analysis of the current Pareto front. The evolutionary process of our approach is divided in 4 stages. Our

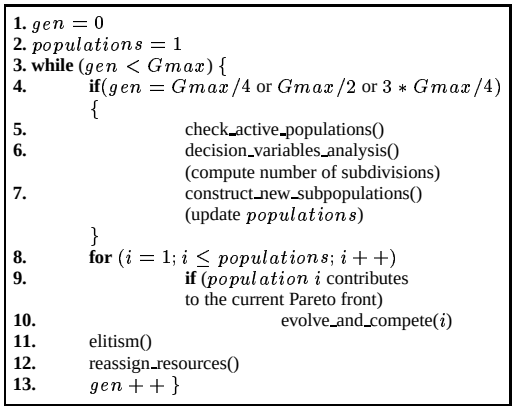

Fig. 1. Pseudocode of our algorithm.

current version equally divides the full evolutionary run into four parts (i.e., the total number of generations is divided by four), and each stage is allocated one of these four parts.

First Stage. During the first stage, the algorithm is allowed to explore all of the search space, by using a population of individuals which are selected using Fonseca and Fleming's Pareto ranking scheme [3]. Additionally, the approach uses the adaptive grid proposed in [4]. At the end of this first stage, the algorithm analyses the current Pareto front (stored in the adaptive grid) in order to determine what variables of the problem are more critical. This analysis consists of looking at the current values of the decision variables corresponding to the current Pareto front (line 6, Figure 1). This analysis is performed independently for each decision variable. The idea is to determine if the values corresponding to a certain variable are distributed along all the allowable interval or if such values are concentrated on a narrower range. When the whole interval is being used, the algorithm concludes that keeping the entire interval for that variable is important. However, if only a narrow portion is being used, then the algorithm will try to identify portions of the interval that can be discarded from the search process. As a result of this analysis, the algorithm determines whether is convenient or not to subdivide (and, in such case, it also determines how many subdivisions to perform) the interval of a certain decision variable. Each of these different regions will be assigned a different population (line 7, Figure 1).

Second Stage. When reaching the second stage, the algorithm consists of a certain number of populations looking each at different regions of the search space. At each generation, the evolution of all the populations takes place independently and, later on, the nondominated elements from each population are sent to the adaptive grid where they "cooperate" and "compete" in order to conform a single Pareto front (line 10, Figure 1). After this, we count the number of individuals that each of the populations contributed to the current Pareto front. Our algorithm is *elitist* (line 11, Figure 1), because after the first generation of the second stage, all the populations that do not provide any individual to the current Pareto front are automatically eliminated and the sizes of the

other populations are properly adjusted. Each population is assigned or removed individuals such that its final size is proportional to its contribution to the current Pareto front. These individuals to be added or removed are randomly generated/chosen. Thus, populations compete with each other to get as many extra individuals as possible. Note that it is, however, possible that the sizes of the populations "converge" to a constant value once their contribution to the current Pareto front does not change any longer.

Third Stage. During the third stage, we perform a check on the current populations in order to determine how many (and which) of them can continue (i.e., those populations which continue contributing individuals to the current Pareto front) (line 5, Figure 1). Over these (presumably good) populations, we will apply the same process from the second stage (i.e., they will be further subdivided and more populations will be created in order to exploit these "promising regions" of the search space). In order to determine the number of subdivisions that are to be used during the third stage, we repeat the same analysis as before. The individuals from the "good" populations are kept. All the good individuals are distributed across the newly generated populations. After the first generation of the third stage, the elitist process takes place and the size of each population will be adjusted based on the same criteria as before. Note however, that we define a minimum population size and this size is enforced for all populations at the beginning of the third stage.

Fourth Stage. During this stage, we apply the same procedure of the third stage in order to allow a fine-grained search.

Decision Variables Analysis. The mechanism adopted for the decision variables analysis is very simple. Given a set of values within an interval, we compute both the minimum average distance of each element with respect to its closest neighbor and the total portion of the interval that is covered by the individuals contained in the current Pareto front. Then, only if the set of values covers less than 80% of the total of the interval, the algorithm considers appropriate to divide it. Once the algorithm decides to divide the interval, the number of divisions gets increased (without exceeding a total of 40 divisions per interval), as explained next. Let's define $range^i$ as the percentage of the total of $interval^i$ that is occupied by the values of the variable i. Let $\bar{d}^i_{min}$ be the minimum average distance between individuals (with respect to the variable i) and let $divisions^i$ be the number of divisions to perform in the interval of the variable i:

if ($range^i$ <0.8∗$interval^i$)
  while ($\bar{d}^i_{min}$ <0.2∗$interval^i$)
  { $divisions^i$ ++; $interval^i$ =0.2∗$interval^i$; }

Parameters Required. Our proposed approach requires the following parameters:

1. Crossover rate (p_c) and mutation rate (p_m).
2. Maximum number of generations ($Gmax$).
3. Size of the initial population ($popsize_{init}$) to be used during the first stage and minimum size of the secondary population ($popsize_{sec}$) to be used during the further stages.

5 Description of the Parallelization Strategy

The topology adopted in this work consisted of a master-slave scheme. As we indicated before, the evolutionary process of our algorithm is divided in **4 stages**. Next, we will briefly describe the part of each of these stages that was parallelized.

First Stage. In the case of Pthreads, this first stage is performed by the master thread. In the case of MPI, the corresponding work is performed by each of the slave processes. Since the master slave is the only one with access to the adaptive grid, upon finishing each generation, each slave process must send its full population to the master process. The master process receives all the populations and applies the corresponding filters to send the nondominated individuals (of each population) to the adaptive grid.

Second Stage. From this stage, the algorithm uses a certain number of populations so that it can explore different regions of the search space. Thus, in the case of Pthreads, given a fixed number of threads, a dynamic distribution of the total number of population takes place: the threads evolve the next available population. At each generation, each thread evolves its corresponding populations and, then, it sends the nondominated individuals from each population to the adaptive grid (line 10, Figure 1). The grid access was implemented with mutual exclusion. After accessing the adaptive grid, the master thread is on charge of counting the number of individuals provided by each population to constitute the current Pareto front, and also on charge of reassigning the resources corresponding to each of the populations (lines 11 and 12, Figure 1). In the case of MPI, given a fixed number of slave processes, we assigned a fixed and equitative number of populations to each process. Once the master process has decided which populations will be assigned to each slave process, it proceeds to transfer them. In order to decrease the sending and/or reception of messages peer-to-peer between processes, we created *buffers*. Thus, each time that one or more full populations need to be sent or received, a buffer is created to pack (or receive) all the necessary information and later on, such information is sent (or unpacked). This is done with all the slaves, such that all can receive their corresponding populations. Finally, each slave sends back all its populations to the master process, such that the master can use them in any procedures required.

Third and Fourth Stages. The main mechanism of these stages, represented by lines 4–7 in Figure 1 is performed by the master thread (process). Then, we continue with the evolutionary process and with the resources reassignment described in the second stage.

Synchronization. In the case of Pthreads, at each generation we must wait until all the threads have finished their corresponding evolutionary processes and grid accesses. Each finished process waits until the continuation signal is received. Once all the processes have finished, the last thread to arrive takes care of the necessary processes and when it finishes, it sends the required signal to awaken all the other threads and continue with the evolutionary process. In the case of MPI, we use barriers at each generation for the synchronization. Such barriers are adopted after sending all the populations to all the slaves and after the reception of all the corresponding populations. This was done such that all the slaves could start the evolutionary process and corresponding evaluations of their individuals at the same time.

6 Comparison of Results

The efficiency of a parallel algorithm tends to be measured in terms of its correctness and its speedup. The speedup (SP) of an algorithm is obtained by dividing the processing time of the serial algorithm (T_s) by the processing time of the parallel version (T_p): $SP = T_s/T_p$. In all the experiments performed, we used the following parameters for our approach: crossover rate (p_c) of 0.8, mutation rate (p_m) of $1/codesize$ and size of the initial population ($popsize_{init}$) equal to the minimum size of the secondary population ($popsize_{sec}$) = 20. The maximum number of generations ($Gmax$) was adjusted such that the algorithms always performed an average of 40000 fitness function evaluations per run. The maximum number of allowable populations was 200. The experiments took place on a PC with 4 processors. In order to give an idea of how good is the performance of the proposed algorithm, we will also include a comparison of results with respect to the Pareto Archived Evolution Strategy (PAES) [4] (PAES was run using the same number of fitness function evaluations as the serial version of our coevolutionary algorithm), which is an algorithm representative of the state-of-the-art in the area. To allow a quantitative comparison of results, the following metrics were adopted:

Error Ratio (ER): This metric was proposed by Van Veldhuizen [7] to indicate the percentage of solutions (from the nondominated vectors found so far) that are not members of the true Pareto optimal set: $ER = (\sum_{i=1}^{n} e_i)/n$ where n is the number of vectors in the current set of nondominated vectors available; e_i = 0 if vector i is a member of the Pareto optimal set, and $e_i = 1$ otherwise.

Inverted Generational Distance (IGD): The concept of generational distance was introduced by Van Veldhuizen & Lamont [8, 9] as a way of estimating how far are the elements in the Pareto front produced by our algorithm from those in the true Pareto front of the problem. This metric is defined as: $GD = (\sqrt{\sum_{i=1}^{n} d_i^2})/n$ where n is the number of nondominated vectors found by the algorithm being analyzed and d_i is the Euclidean distance (measured in objective space) between each of these and the nearest member of the true Pareto front. In our case, we implemented an "inverted" generational distance metric (IGD) in which we use as a reference the true Pareto front, and we compare each of its elements with respect to the front produced by an algorithm.

For each of the examples shown below, we performed 30 runs per algorithm. The Pareto fronts that we will show correspond to the median of the 30 runs performed with respect to the ER metric.

6.1 Test Function 1

$$\begin{aligned} \text{Min } f_1(x_1, x_2) = x_1, \quad \text{Min } f_2(x_1, x_2) &= (1.0 + 10.0x_2) \\ (1.0 - \frac{x_1^2}{1.0 + 10.0x_2} &- \frac{x_1}{1.0 + 10.0x_2} \sin(2\pi 4x_1)) \\ 0.0 \leq x_1, x_2 &\leq 1.0 \end{aligned} \tag{4}$$

Table 1 shows the values of SP and the metrics ER and IGD for each of the versions compared.

		PAES	Serial	Pthreads	MPI
				SP=2.9541	SP=2.5185
ER	best	0.01	0.22	0.16	0.09
	median	0.06	0.40	0.36	0.39
	worst	0.12	0.57	0.57	0.56
	average	0.057	0.39	0.37	0.36
	std. dev.	0.0301	0.1164	0.1215	0.1283
IGD	best	0.001030	0.000596	0.000606	0.000564
	median	0.001305	0.000818	0.000807	0.000875
	worst	0.003224	0.003277	0.003277	0.002906
	average	0.001382	0.001062	0.001061	0.001204
	std. dev.	0.000409	0.000638	0.000638	0.000637

Table 1. Comparison of results for the first test function. SP refers to the speedup achieved.

6.2 Test Function 2

$$\text{Min } f_1(x_1, x_2) = x_1, \quad \text{Min } f_2(x_1, x_2) = \frac{g(x_2)}{x_1}$$

$$g(x_2) = 2.0 - e^{-(\frac{x_2 - 0.2}{0.004})^2} - 0.8e^{-(\frac{x_2 - 0.6}{0.4})^2}, \quad 0.1 \leq x_1, x_2 \leq 1.0$$

Table 2 shows the values of SP and the metrics ER and IGD for each of the versions compared.

		PAES	Serial	Pthreads	MPI
				SP=2.2486	SP=2.5639
ER	best	0.01	0.06	0.03	0.07
	median	0.11	0.17	0.16	0.16
	worst	1.0	0.56	0.56	0.77
	average	0.26	0.14	0.14	0.14
	std. dev.	0.3390	0.1191	0.1197	0.1842
IGD	best	0.005430	0.002321	0.002611	0.002882
	median	0.009875	0.003642	0.003648	0.003444
	worst	0.023626	0.007876	0.007876	0.010387
	average	0.010495	0.003747	0.003791	0.003914
	std. dev.	0.004310	0.001035	0.000984	0.001431

Table 2. Comparison of results for the second test function. SP refers to the speedup achieved.

7 Discussion of Results

In the case of the first test function, when comparing results with respect to the ER metric, PAES considerably improves the results achieved by our coevolutionary approach in all of its versions. However, note that with respect to the IGD metric, our approach presents a better average performance than PAES. On the other hand, it is very important to note that, despite the fact that the use of Pthreads improves the results of the serial version (with respect to the ER metric), the MPI version does even better. With

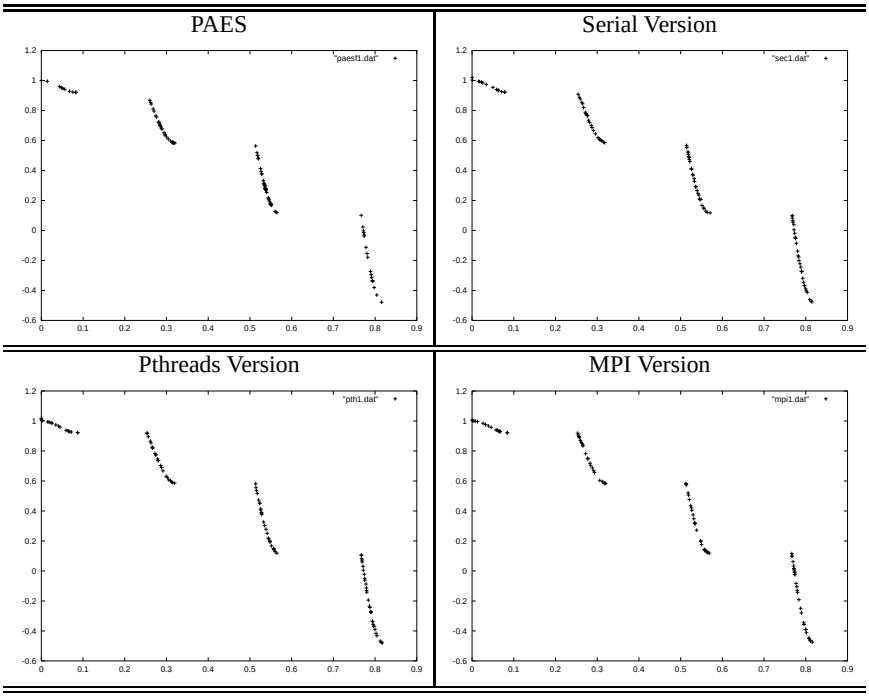

Fig. 2. Pareto fronts obtained by the serial versions of PAES and our coevolutionary algorithm, the Pthreads version and the MPI version for the first test function.

respect to the IGD metric, the three versions of the coevolutionary algorithm have a similar behavior in all the test functions. In general, the results when using Pthreads are very similar than those obtained with the serial version. In contrast, the MPI version produces (marginally) better results than the other two versions in the case of the first function, and (marginally) poorer results in the case of the second function.

In the second test function, our coevolutionary approach had a better average performance than PAES with respect to the ER metric. In fact, note that the worst solutions achieved by PAES totally missed the true Pareto front of the problem (therefore the value of 1.0 obtained). Regarding the IGD metric, the results of the three versions of our coevolutionary algorithm are approximately three times better than those obtained by PAES. In the case of the second test function, the mere use of Pthreads improves on the results obtained using the serial version of the algorithm (when measured with respect to the ER metric). On average, however, the results obtained by our MPI approach are of the same quality as those obtained with the serial version of the algorithm.

In general, regarding the ER metric, the four algorithms reach the true Pareto front of each problem, and the first test function is the only one in which PAES is found to be superior to our approach (in any of its versions). However, regarding the IGD metric, we can see that the results of our coevolutionary algorithm are better than those obtained by PAES. Graphically, we can see that this is due to the fact that PAES has problems

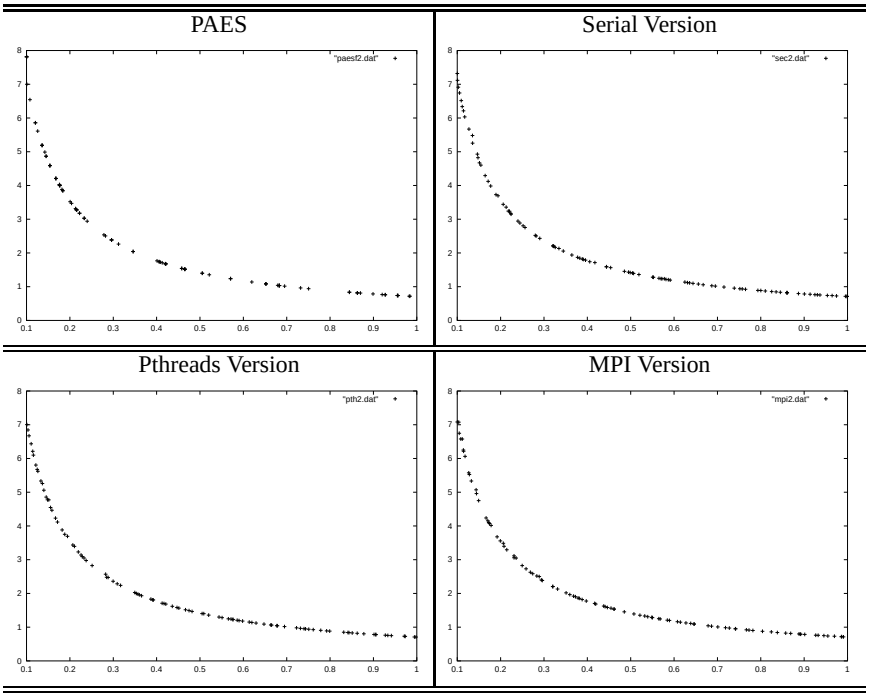

Fig. 3. Pareto fronts obtained by the serial versions of PAES and our coevolutionary algorithm, the Pthreads version and the MPI version for the second test function.

to cover the entire Pareto front of the problem. Regarding the speedups achieved, in the first test function, the Pthreads implementation was superior to the MPI version. In the second example, the best speedup was achieved by our MPI strategy. The speedup values that we obtained are considered acceptable if we take into account that the parallelization strategy adopted in this study is rather simple and does not adopt the best possible workload for the 4 processors available.

8 Conclusions and Future Work

We presented a simple parallelization of a coevolutionary multi-objective optimization algorithm. The main idea of our algorithm is to obtain information along the evolutionary process as to subdivide the search space into subregions, and then to use a subpopulation for each of these subregions. At each generation, these different subpopulations "cooperate" and "compete" among themselves and from these different processes we obtain a single Pareto front. The size of each subpopulation is adjusted based on their contribution to the current Pareto front. Thus, those populations contributing with more nondominated individuals have a higher reproduction probability. Three versions of the algorithm were compared in this paper: the serial version and two parallel versions:

one using Pthreads and another one using MPI. We also included a comparison of results with respect to PAES to have an idea of the performance of the serial version of our algorithm. The results obtained indicate that, despite the simplicity of the parallel strategy that we implemented, the gains in execution time are considerably good, without affecting (in a significant way) the quality of the results with respect to the serial version. As part of our future work we are considering the use of a more efficient parallelization strategy that can improve the $speedup$ values achieved in this paper. We are also considering certain structural engineering applications for our proposed approach.

Acknowledgments

The first author acknowledges support from CONACyT through project number 34201-A. The second author acknowledges support from CONACyT through a scholarship to pursue graduate studies at CINVESTAV-IPN's Electrical Engineering Department.

References

1. Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers, New York (2002) ISBN 0-3064-6762-3.
2. Coello Coello, C.A., Reyes Sierra, M.: A Coevolutionary Multi-Objective Evolutionary Algorithm. In: Proceedings of the Congress on Evolutionary Computation, IEEE Press (2003) (accepted for publication)
3. Fonseca, C.M., Fleming, P.J.: Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In Forrest, S., ed.: Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, California, University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers (1993) 416–423
4. Knowles, J.D., Corne, D.W.: Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. Evolutionary Computation **8** (2000) 149–172
5. Paredis, J.: Coevolutionary algorithms. In Bäck, T., Fogel, D.B., Michalewicz, Z., eds.: The Handbook of Evolutionary Computation, 1st supplement. Institute of Physics Publishing and Oxford University Press (1998) 225–238
6. Potter, M., Jong., K.D.: A cooperative coevolutionary approach to function optimization. In: Proceedings from the Fifth Parallel Problem Solving from Nature, Jerusalem, Israel, Springer-Verlag (1994) 530–539
7. Van Veldhuizen, D.A.: Multiobjective Evolutionary Algorithms: Classifi cations, Analyses, and New Innovations. PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio (1999)
8. Van Veldhuizen, D.A., Lamont, G.B.: Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio (1998)
9. Van Veldhuizen, D.A., Lamont, G.B.: On Measuring Multiobjective Evolutionary Algorithm Performance. In: 2000 Congress on Evolutionary Computation. Volume 1., Piscataway, New Jersey, IEEE Service Center (2000) 204–211