MOGA-II PERFORMANCE ON NOISY OPTIMIZATION PROBLEMS

Silvia Poles
ES.TEC.O S.r.l.
Trieste, Italy
poles@esteco.com

Enrico Rigoni
ES.TEC.O S.r.l.
Trieste, Italy
rigoni@esteco.com

Tea Robič
Department of Intelligent Systems
Jožef Stefan Institute, Ljubljana, Slovenia
tea.robic@ijs.si

Abstract Since the mid-fifties evolutionary algorithms (EAs) have been used in different optimization problems. In the last years their use was extended to the demanding field of multi-objective optimization. For this expansion, EAs themselves had to evolve to more complex forms. The question is whether an algorithm that is adapted to work well with multiple-objectives is still capable to handle single-objective optimization problems. In this paper we present a new EA for multi-objective optimization called MOGA-II. We test it on noisy single-objective problems and compare its performance with two algorithms for single-objective optimization. The results show that MOGA-II is a robust algorithm that can efficiently solve a palette of different optimization problems.

Keywords: MOGA-II, Genetic algorithms, Optimization, Noisy functions

1. Introduction

Evolutionary Algorithms (EAs) are widely used in several optimization problems that are too complex to be solved by traditional methods such as linear programming or gradient based algorithms. Their main advantage over traditional methods is robustness, which enables efficient optimization of different functions, including noisy ones. Being population-based, EAs can be easily parallelized and can construct multiple optimal solutions in a single run. The latter property is especially welcome in multimodal and multi-objective optimization.

Multi-objective Optimization Problems (MOPs) are more complex than Single-objective Optimization Problems (SOPs) because they involve optimization of several (usually conflicting) objectives. This yields not a single optimal solution but a set of equally important optima, called the *Pareto front*. In multi-objective optimization it is important to guide the search process toward the Pareto front and at the same time maintain adequate population variety to capture as many diverse optimal solutions as possible.

In recent years, EAs have been adjusted in numerous approaches to handle MOPs. Among many algorithms, the NSGA-II of Deb et al. [1] and SPEA2 of Zitzler et al. [10] are the most popular. One of the new EAs for multi-objective optimization is MOGA-II described by Poles in [6], which uses a directional crossover operator for fast convergence and a smart multisearch elitism for uniform spread of solutions. MOGA-II has proved to be very efficient in solving MOPs [7] and in this paper it is tested for robustness. We are raising the question: "Can an EA that was constructed to solve MOPs handle also (noisy) SOPs?" For this purpose we test MOGA-II on five benchmark problems and compare its results with the ones obtained by differential evolution (DE) [8] and a standard EA for single-objective optimization.

The rest of the paper is organized as follows: a detailed description of MOGA-II is presented in Sect. 2 followed by the specification of the experiments in Sect. 3. Section 4 is devoted to the presentation of the experimental results that are further discussed in Sect. 5. The paper ends with a conclusion in Sect. 6.

2. MOGA-II

MOGA-II is an improved version of MOGA (Multi-Objective Genetic Algorithm) by Poloni [5] and is not to be confused with MOGA by Fonseca and Fleming [2] with which it shares only the same acronym. MOGA-II uses a smart multisearch elitism for robustness and directional crossover for fast convergence. Its efficiency is ruled by its operators (classical crossover, directional crossover, mutation and selection) and by the use of elitism. In this paper only the features of MOGA-II that relate to its use in single-objective optimization are explained.

2.1 Encoding

Encoding in MOGA-II is done as in classical genetic algorithms [3]. Each variable is represented as a binary string where the length of the string depends on the base (the number of allowed values for the variable). For example, if only integer values in the interval $[0, 10]$ are to be allowed (11 possible values), the base is set to 11. Thus the length of the string is equal to 4 and the variable can take values from $[0000]$ to $[1011]$. In order to simulate a continuous variable, the base must be set to an appropriate high number.

2.2 Elitism

Elitism is very important in multi-objective optimization because it helps preserving the individuals that are closest to the Pareto front and the ones that have the best dispersion. When optimizing a single objective, the elitism embedded in MOGA-II reduces to copying the solution with the best fitness into the next generation.

2.3 Reproduction

MOGA-II uses four different operators for reproduction (one-point crossover, directional crossover, mutation and selection). At each step of the reproduction process, one of the four operators is chosen (with regard to the predefined operator probabilities) and applied to the current individual. Algorithm 1 shows the reproduction of MOGA in pseudo code.

Algorithm 1: Pseudo code of the reproduction used in MOGA-II

```
with (individual $Ind_i \in$ generation $G$) do
  choose reproduction operator
  if (operator is one-point crossover) then
    $j \leftarrow$ TournamentSelection, where $j \neq i$
    $NewInd_i \leftarrow$ OnePointCrossover($Ind_i$, $Ind_j$)
  else if (operator is directional crossover) then
    $j \leftarrow$ RandomWalk($i$)
    $k \leftarrow$ RandomWalk($i$), where $k \neq j \neq i$
    $NewInd_i \leftarrow$ DirectionalCrossover($Ind_i$, $Ind_j$, $Ind_k$)
  else if (operator is mutation) then
    $NewInd_i \leftarrow$ Mutation($Ind_i$)
  else if (operator is selection) then
    $NewInd_i \leftarrow Ind_i$
  end if
end with
```

2.3.1 One-Point Crossover. One-point crossover is the most classical operator for reproduction. Two parents are chosen and some portion of the genetic material (the design variables) is exchanged between the parent variables vectors (see Fig. 1). The point of the crossing site is randomly chosen and the binary strings are cut at that point. The two head pieces are then swapped and rejoined with the two tail pieces. From the resulting individuals, usually called children, one is randomly selected to be the new individual.

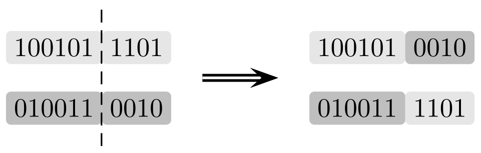

Figure 1. One-point crossover.

In MOGA-II, one-point crossover starts by taking the current individual Ind_i as the first parent. The second parent Ind_j is chosen by means of a multi-objective tournament selection on a randomly selected population subset: this operator returns the first non-dominated solution in the subset.

2.3.2 Directional Crossover. Directional crossover is slightly different and assumes that a *direction of improvement* can be detected comparing the fitness values of two reference individuals. In [9] a novel operator called *evolutionary direction crossover* was introduced and it was shown that even in the case of a complex multimodal function this operator outperforms classical crossover.

The direction of improvement is evaluated by comparing the fitness of the individual Ind_i from generation t with the fitness of its parents belonging to generation $t-1$. The new individual is then created by moving in a randomly weighted direction that lies within the ones individuated by the given individual and his parents (see Fig. 2). A similar concept can be however applied on

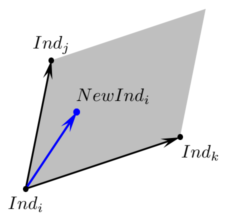

Figure 2. Directional crossover between individuals Ind_i, Ind_j and Ind_k.

the basis of directions not necessarily linked to the evolution but detected by selecting two other individuals Ind_j and Ind_k in the same generation (like shown in Algorithm 1).

Algorithm 2: Random walk from the i-th individual

```
Input: index $i$ of the starting individual
  $S \leftarrow \emptyset$
  $m \leftarrow \lfloor \sqrt{popSize} \rfloor$;
  for all (N steps) do
    $k \leftarrow \lfloor 4 \cdot rand() + 1 \rfloor$
    if ($k == 1$) then
      $i \leftarrow i + 1$
    end if
    if ($k == 2$) then
      $i \leftarrow i - 1$
    end if
    if ($k == 3$) then
      $i \leftarrow i - m$
    end if
    if ($k == 4$) then
      $i \leftarrow i + m$
    end if
    if ($i < 1$) then
      $i \leftarrow i + popSize$
    end if
    if ($i > popSize$) then
      $i \leftarrow i - popSize$
    end if
    $S \leftarrow S \cup Ind_i$
  end for
Output: $j$ such that $f(Ind_j) = \min_{Ind \in S} f(Ind)$
```

The selection of individuals Ind_j and Ind_k can be done using any available selection schema. In MOGA-II local tournament with random steps in a toroidal grid is used. First of all, the individual subject to reproduction is chosen as the starting point. Other individuals met in a random walk of assigned number of steps from that starting point are then marked as possible candidates for the first "parent" Ind_j. The list of all possible candidates for the second "parent" Ind_k is selected in the same way in a successive (and generally different) random walk from the same starting point. When the set of candidates is generated, the candidate with the best fitness is chosen.

The number of steps N in the random walk remains fixed during the entire optimization run and is proportional to the population size. Algorithm 2 shows the random walk with individual Ind_i chosen as a starting point. The function *rand()* generates values in the interval $[0, 1)$ with a uniform distribution.

Directional crossover has demonstrated to help the algorithm convergence for a wide range of numerical problems.

2.3.3 Mutation. Mutation is an operator that ensures *diversity* from one generation to the next. Using plain words we can say that mutation guarantees the algorithm robustness. In MOGA-II it is possible to define the value of the so-called *DNA String Mutation Ratio*. This value gives the percentage of the binary string that is perturbed by the mutation operator.

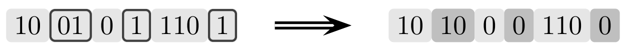

Figure 3. Mutation example with DNA string mutation ratio set to 40%.

3. Experiments

In our experiments we tested the performance of MOGA-II on five numerical single-optimization problems with and without noise. The used numerical benchmark problems are described in Table 1. In all problems the function is to be minimized.

To preserve consistency with the results in [4] we set up the following experiments. Each test function f_i was optimized with and without noise. The noise was introduced as

$$f_i^*(\mathbf{x}) = f_i(\mathbf{x}) + N(0,1)$$

where the $N(0,1)$ is the normal (or Gaussian) distribution with mean 0 and variance 1. The experiments on the noisy functions were run with 5 different number of resamples: s = 1, 5, 20, 50 and 100. This means that a solution was evaluated s times and the true value of f_i was estimated by the mean of the samples. In all runs the number of function evaluations was kept constant to provide a fair performance comparison and was calculated as

$$numEval = popSize \times numIt \times s - numUnchanged$$

where $popSize$ is the population size, $numIt$ is the number of iterations, s is the number of resamples and $numUnchanged$ is the number of unchanged individuals during the run. The latter refers to candidate solutions that were evaluated previously in the same run, which we did not re-evaluate if they remained unchanged, such as for example members of the elite.

Each experiment was repeated 3 times. We used $numEval = 100,000$ for low dimensional functions f_1 and f_2 and $400,000$ for the 50 dimensional

Table 1. Benchmark problems.

name	Schaffer F6
dimensions	2
definition	$f_1(\mathbf{x}) = 0.5 + \frac{\sin^2(\sqrt{x_1^2+x_2^2})-0.5}{(1+0.001(x_1^2+x_2^2))^2}$
constraints	$\mathbf{x_i} \in [-100, 100]$
optimum	$\mathbf{x}^* = (0,0)$, $f_1(\mathbf{x}^*) = 0$
name	Sphere
dimensions	5
definition	$f_2(\mathbf{x}) = \sum_{i=1}^{5} x_i^2$
constraints	$\mathbf{x_i} \in [-100, 100]$
optimum	$\mathbf{x}^* = (0,0,0,0,0)$, $f_2(\mathbf{x}^*) = 0$
name	Griewank
dimensions	50
definition	$f_3(\mathbf{x}) = 1 + \frac{1}{4000} \cdot \sum_{i=1}^{50}(x_i - 100)^2 - \prod_{i=1}^{50} \cos\left(\frac{x_i - 100}{\sqrt{i}}\right)$
constraints	$\mathbf{x_i} \in [-600, 600]$
optimum	$\mathbf{x}^* = (100, \ldots, 100)$, $f_3(\mathbf{x}^*) = 0$
name	Rastrigin F1
dimensions	50
definition	$f_4(\mathbf{x}) = 500 + \sum_{i=1}^{50}(x_i^2 - 10 \cdot \cos(2\pi x_i))$
constraints	$\mathbf{x_i} \in [-5.12, 5.12]$
optimum	$\mathbf{x}^* = (0, \ldots, 0)$, $f_4(\mathbf{x}^*) = 0$
name	Rosenbrock
dimensions	50
definition	$f_5(\mathbf{x}) = \sum_{i=1}^{49}\left(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right)$
constraints	$\mathbf{x_i} \in [-50, 50]$
optimum	$\mathbf{x}^* = (1, \ldots, 1)$, $f_5(\mathbf{x}^*) = 0$

Table 2. The values of population size and number of iterations for each experiment. The number of unchanged individuals is different in every run, therefore the number of iterations is calculated as $numEval/(popSize \times s)$. The non-noisy versions of the functions had the same values for $popSize$ and $numIt$ as the respective noisy ones with $s = 1$.

functions	s	1	5	20	50	100
f_1^*, f_2^*	$popSize$	50	50	50	50	25
	$numIt$	2000	400	100	40	40
f_3^*, f_4^*, f_5^*	$popSize$	100	100	100	100	100
	$numIt$	4000	800	200	80	40

functions f_3, f_4 and f_5. The values of $popSize$ and $numIt$ for each experiment are gathered in Table 2.

Table 3 shows the parameter setting for MOGA-II. The parameters were not tuned to any benchmark problem – we used the default values of the algorithm, which demonstrate to perform well in most real-world problems. MOGA-II encodes its variables as binary strings and therefore needs a discretized space. In this experiments, the base for the encoding was set in such way, that the discretization was 10^{-6}. In the used benchmark problems, the bases have been set respectively to $200,000,001$ for f_1 and f_2, $1,200,000,001$ for f_3, $10,240,001$ for f_4 and $100,000,001$ for f_5. As said in the previous section, such high bases permit to simulate suitably continuous variables.

Table 3. Parameter settings for MOGA-II.

probability of directional crossover	50%
probability of classical crossover	35%
probability of selection	5%
probability of mutation	10%
DNA string mutation ratio	5%

4. Results

The results reported in this section refer to the "true" (non-noisy) evaluation of solutions. We compare the results of MOGA-II with the ones obtained by differential evolution and a standard EA that were published in [4] (see Table 4). The comparison is not completely fair. As said in the previous section, the experiments with MOGA-II were repeated 3 times while the results of DE and EA from Table 4 are the outcome of 30 runs. Moreover, with the high dimensional functions f_3, f_4 and f_5 MOGA-II was given 400,000 evaluations while DE and EA continued until 500,000 evaluations were reached. The reason for this inconsistence is the lack of time we had at our disposal. But it is important to emphasize that this comparison works in favour of DE and EA and not MOGA-II. Time limitations were due to the fact that MOGA-II is embedded into a commercial optimization environment called modeFRONTIER [11]; this Java software contains an advanced post-processing analysis tool, well suited for engineering issues, where the bottleneck is given by the external solvers. But in our case the main problem was handling such a big designs database, and this cannot be quickly done in a non-compiled programming language such as Java.

In Fig. 4 the charts represent the performance of MOGA-II on noisy benchmark problems. Note that there was not enough room to include also the graphs for non-noisy evaluations. Anyhow, with non-noisy functions the optimum was always reached very quickly.

Table 4. Mean and standard deviation of the final results for the benchmark problems (see Table 1). The results of the algorithms DE and EA are taken from [4].

	MOGA-II		*DE*		*EA*	
Function	*mean*	*st. dev.*	*mean*	*st. dev.*	*mean*	*st. dev.*
f_1	0	0	0	0	$3 \cdot 10^{-17}$	0
f_1^* $(s = 1)$	0.04285	0.03488	0.48998	0.00582	0.25829	0.03045
f_1^* $(s = 5)$	0.03533	0.04419	0.40360	0.03030	0.12859	0.01678
f_1^* $(s = 20)$	0.01054	0.00120	0.16597	0.02753	0.06730	0.01066
f_1^* $(s = 50)$	0.01012	0.00049	0.12729	0.01829	0.04769	0.00757
f_1^* $(s = 100)$	0.01927	0.01655	0.09795	0.01203	0.06277	0.00743
f_2	0	0	10^{-152}	0	$7 \cdot 10^{-20}$	0
f_2^* $(s = 1)$	0.01067	0.00581	0.25249	0.02603	0.04078	0.00543
f_2^* $(s = 5)$	0.01077	0.00188	0.13315	0.01266	0.02690	0.00363
f_2^* $(s = 20)$	0.00757	0.00903	0.07364	0.00811	0.02205	0.00290
f_2^* $(s = 50)$	0.00398	0.00238	0.07004	0.00686	0.01765	0.00233
f_2^* $(s = 100)$	0.04436	0.04290	0.08165	0.00800	0.03929	0.00396
f_3	$2 \cdot 10^{-12}$	$2 \cdot 10^{-12}$	0	0	0.00624	0.00138
f_3^* $(s = 1)$	3.29905	0.40864	3.31514	0.07388	1.14598	0.00307
f_3^* $(s = 5)$	2.40897	0.25458	2.42183	0.03616	1.10223	0.00342
f_3^* $(s = 20)$	1.78540	0.35279	2.67093	0.03895	1.44349	0.01381
f_3^* $(s = 50)$	3.78713	0.75639	46.8197	0.96449	3.69626	0.13127
f_3^* $(s = 100)$	14.5960	1.25293	233.802	6.25840	18.0858	0.99646
f_4	0.49748	0.70354	0	0	32.6679	1.94017
f_4^* $(s = 1)$	28.8315	2.33133	2.35249	0.06062	30.7511	1.32780
f_4^* $(s = 5)$	21.6573	2.68711	14.0355	0.47935	31.4725	2.02356
f_4^* $(s = 20)$	45.2687	4.17703	167.628	2.12569	39.1777	2.11529
f_4^* $(s = 50)$	104.415	19.7481	314.762	2.88650	74.8577	2.69437
f_4^* $(s = 100)$	177.847	13.7495	438.036	3.67504	147.800	2.93208
f_5	40.6641	50.5749	35.3176	0.27444	79.8180	10.4477
f_5^* $(s = 1)$	56.5750	39.8582	47.6188	0.15811	118.940	13.2322
f_5^* $(s = 5)$	160.737	56.7030	47.0404	0.13932	341.788	49.6738
f_5^* $(s = 20)$	1601.84	1081.10	7917.46	352.851	1859.06	261.844
f_5^* $(s = 50)$	$1.3 \cdot 10^5$	93260.1	$1.7 \cdot 10^7$	903677	35477.7	4656.17
f_5^* $(s = 100)$	$1.2 \cdot 10^6$	$4.3 \cdot 10^5$	$3.0 \cdot 10^8$	$1.0 \cdot 10^7$	257488	19371.2

5. Discussion

The results in the non-noisy cases are obviously better than the noisy experiments: the noise plays always the role of annoyance factor. As concerns the noisy cases, considering the different values $s = 1$, 5, 20, 50, and 100 for the resampling (and consequently the differences in the number of iterations and/or population size, in order to preserve the total number of evaluated designs), we can expect two different effects to come into play, for “low” and “high” values of s, respectively. Towards the low end, say $s = 1$, the noise gain in importance, so we could expect the results to deteriorate. But also towards the high end, i.e. $s = 100$, the results are expected to be worse, since the request for evaluating

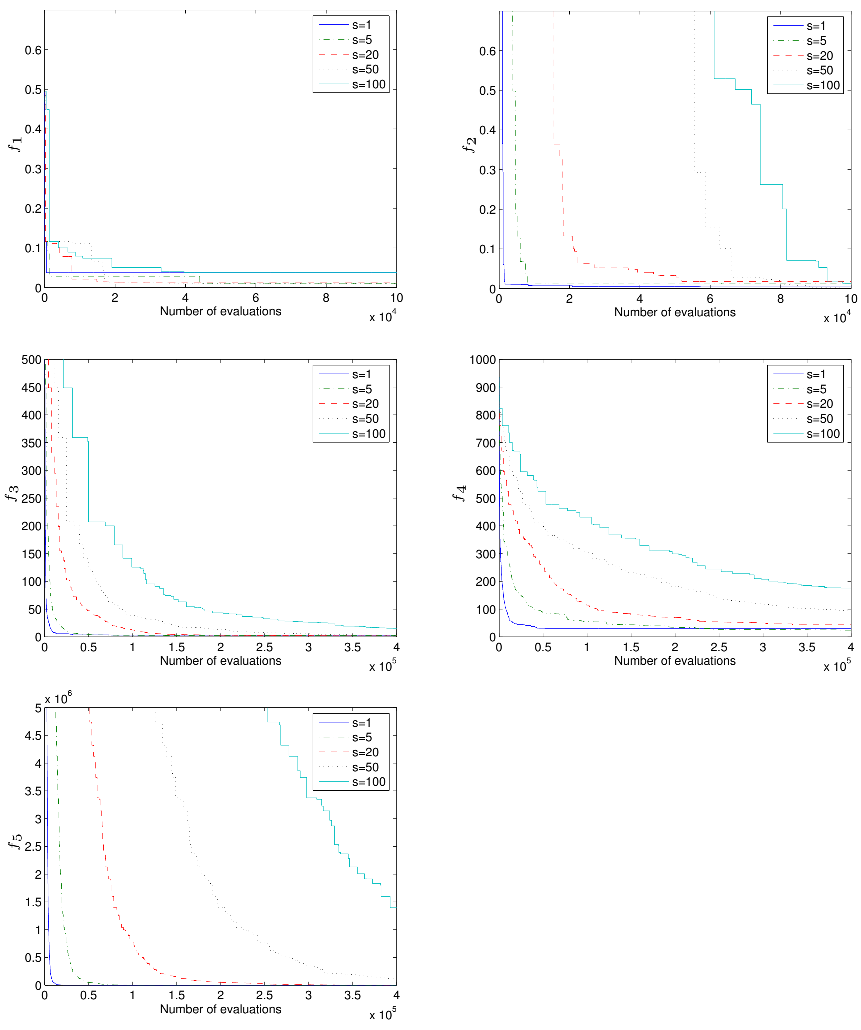

Figure 4. MOGA-II performance on noisy benchmark problems (see Table 1 for function definitions).

the same candidate solution many times, in order to reduce the noise effect, implies that we have to limit consequently the number of iterations, stopping prematurely the optimization process. This is the case for the low-dimensional noisy Schaffer F6 (2D) function (f_1) and Sphere (5D) function (f_2), but also for the high-dimensional Griewank (50D) function (f_3): the best results are found for $s = 20$ or $s = 50$, as a compromise between the low and high ends.

On the contrary, for the Rosenbrock (50D) function (f_5), the results get better monotonically as s decreases, showing no deterioration due to a stronger noise effect. For Rastrigin F1 (50D) function (f_4) the behaviour is similar, but there is still a residual “low end effect”, which settles the best compromise result in $s = 5$. In these two cases, as outlined by Krink et al. [4], the main problem resides in the difficulty of the function, and not in the noise effect: the performance of any EA is affected by the intrinsic critical aspects of the function, well before the noise effect can come into play, in terms of fitness contribution. This can be also seen considering the results for the non-noisy cases: the results achieved for f_4 and especially for f_5 are sensibly far from the optimal value 0.

The comparison of MOGA-II results with those of DE and the generic EA presented in [4], shows the good performance of MOGA-II (see Table 4). For both low-dimensional noisy functions f_1 and f_2 , MOGA-II performs better than DE and EA, for all values of s.

As concerns f_3, where EA is better than DE, for low s, i.e. $s = 1$ and $s = 5$, MOGA-II is comparable to DE, while for high s, i.e. $s = 20$, 50, and 100 it is comparable to the good results of EA. With the f_4 function, for low s (i.e. $s = 1$, 5), where DE performs better than EA, MOGA-II results are better than EA but worse than DE; for high s (i.e. $s = 20$, 50, and 100), where EA is better than DE, the MOGA-II results are roughly comparable to those of EA. Finally, as concerns f_5, for low s DE performs better than EA, and conversely for high s EA is better than DE, as in the previous case. The results of MOGA-II are roughly situated in an intermediate position between the results of DE and EA, but the large standard deviations prevent us from a detailed comparative analysis. Such large standard deviations could be indicative of a difficult problem, but can also be due to the low number of repetitions. Unfortunately, as said in the previous section, we could repeat the experiments only 3 times, for intrinsic limitations.

It should be noted that in general MOGA-II converges to the optimal solution faster than DE, in terms of number of function evaluations: this is an important point in case of realistic applications, where the evaluation time for a single design can be very long.

6. Conclusion

In this paper, we have presented MOGA-II, a new evolutionary algorithm for multi-objective optimization, and tested it on single-objective optimization problems (with and without noise). Although compared to two very successful algorithms (differential evolution and a standard evolutionary algorithm), MOGA-II sometimes performed better and never worse than both algorithms, which were constructed for single-objective optimization.

The current results motivate further work in testing MOGA-II, especially to see its performance on real-world problems. We can conclude that MOGA-II is a competent algorithm that is robust and efficient enough to handle different optimization problems, including noisy ones.

References

[1] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the 6-th International Conference Parallel Problem Solving from Nature (PPSN-VI)*, 2000, pp. 849–858.

[2] C.M. Fonseca and P.J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, USA, 1993, pp. 416–423.

[3] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, Mass., USA, 1989.

[4] T. Krink, B. Filipič, G.B. Fogel, and R. Thomsen. Noisy optimization problems - A particular challenge for differential evolution? In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, Portland, USA, 2004, pp. 332–339.

[5] C. Poloni and V. Pediroda. GA coupled with computationally expensive simulations: tools to improve efficiency. In *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, John Wiley and Sons, England, 1997, pp. 267–288.

[6] S. Poles. MOGA-II An Improved Multi-Objective Genetic Algorithm. Technical report 2003-006, Esteco, Trieste, 2003.

[7] S. Poles. Bench-marking MOGA-II. Technical report 2004-001, Esteco, Trieste, 2004.

[8] K.V. Price and R. Storn. Differential evolution – a simple evolution strategy for fast optimization. *Dr. Dobb's Journal*, 22(4):18–24, 1997.

[9] K. Yamamoto and O. Inoue. New evolutionary direction operator for genetic algorithms. *AIAA Journal*, 33:1990–1993, 1995.

[10] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. TIK-Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zürich, Switzerland, May 2001.

[11] Web page of the modeFRONTIER optimization software (http://www.esteco.com/Products/modeFrontier/index.html).