New Approaches to Coevolutionary Worst-Case Optimization

Jürgen Branke and Johanna Rosenbusch

Institute AIFB, University of Karlsruhe, Germany
`branke@aifb.uni-karlsruhe.de, johanna.rosenbusch@student.kit.edu`

Abstract. Many real-world optimization problems involve uncertainty. In this paper, we consider the case of worst-case optimization, i.e., the user is interested in a solution's performance in the worst case only. If the number of possible scenarios is large, it is an optimization problem by itself to determine a solution's worst case performance. In this paper, we apply coevolutionary algorithms to co-evolve the worst case test cases along with the solution candidates. We propose a number of new variants of coevolutionary algorithms, and show that these techniques outperform previously proposed coevolutionary worst-case optimizers on some simple test problems.

1 Introduction

Many real life optimization problems involve some form of uncertainty, e.g., because they rely on forecasts, because they depend on an opponent's move, or because the solution eventually implemented is subject to manufacturing tolerances. In such cases, one typically searches for a *robust* solution. Often used criteria are a good expected quality or a low variance (see, e.g., (3, p. 127)). In the following, we consider the case that a user is interested only in a solution's worst-case performance, for example, because the application may include the risk of very severe consequences, such as death or bankruptcy.

Possible applications of worst-case optimization include engineering design (12), portfolio management (10) and scheduling (2, 6, 8)). There exist several ways to approach worst-case optimization, e.g., the calculation of reliability (4) or the use of an embedded EA to identify, for each solution, the worst-case (1). The latter implies great computational efforts which may render the approach infeasible in practice. An alternative and more efficient way to search for a robust solution *and* for its worst case simultaneously is provided by coevolutionary algorithms.

2 Coevolutionary Worst-Case Optimization

There exist several forms of coevolutionary algorithms (CEAs) but we consider only competitive, test-based CEAs which comprise one population consisting of solution candidates and one population forming the test cases (see, e.g., (7, 13, 14) for some early work).

CEAs offer several amenities. They do not need an objective, external metric to evaluate the solutions. Instead, individuals are evaluated by letting them interact with each other. Therefore, CEAs are applicable to problems where an objective criterion does not exist or cannot be computed. This is also the case in worst-case optimization, since the worst-case scenario is unknown, and a test with all possible scenarios may be impossible. CEAs are more efficient in that they use only a limited number of test cases to evaluate a solution. The population of test cases is furthermore selected *adaptively* because it coevolves with the solution candidates and therefore increases in difficulty as the solutions grow more powerful (5).

Over the course of research and application, CEAs have also shown some shortcomings. The first is a direct consequence of the lack of an objective metric: the real (objective) quality of a solution does not necessarily correspond to the subjective quality, i.e., the quality perceived by the algorithm. Furthermore, CEAs are susceptible to various pathologies such as evolutionary forgetting, cycling, disengagement, or overspecialization. For a detailed analysis of these pathologies as well as possible remedies, please refer to, e.g., (11).

CEAs have been applied to a wide range of problems but only few involve worst-case optimization. In (15) a so-called "nested minimax optimization" is performed. One population consists of various designs for a neural controller. The other population persists of plants, i.e., the scenarios in which the controller will be utilized. (2) uses a CEA to solve constrained optimization problems, written as min-max problems. One population evolves the parameter which is responsible for the minimization, the other population represents the parameter which maximizes. (6) and (8) apply worst-case CEAs to the area of scheduling, trying to find robust schedules. One population evolves the schedules, the other population evolves difficult problem instances or possible machine failures. Furthermore, (9) deals with the topic of worst-case optimization on a more theoretical level. We will discuss the basic idea of fitness assignment in these approaches in Section 4.

3 Coevolutionary Algorithm and Test Problems

The basic coevolutionary algorithm considered here uses two populations P_S and P_T. The solutions $s \in P_S$ attempt to minimize a function $F(s,t)$, while the test cases $t \in P_T$ are responsible for identifying the worst cases (i.e., attempt to maximize $F(s,t)$). Each individual is a single real number, and a standard $(\mu + \lambda)$ evolution strategy is used on both populations with mutation as the only variation operator. The mutation operator is additive Gaussian, and mutation probability is 100%.

We test the methods on two different functions. In each function, the solutions, denoted as s, aim at minimizing $F(s,t)$ while the test cases, t, aim at maximizing $F(s,t)$. The functions were designed in such way that the worst case is $t = s$ and the optimum solution is $s = 0$.

$$F1(s,t) = 2st - t^2, \qquad s \in [-50;50], t \in [-50;50] \tag{1}$$

The optimum of function $F1$ is stable because the solution candidates have no incentive to deviate from $(0,0)$.

$$F2(s,t) = s^2 - s\cos(3(s - \frac{\pi}{2})) - (3|s-t| - (s-t)\cos(3((s-t) - \frac{\pi}{2}))),$$
$$s \in [-10;10], t \in [-10;10] \tag{2}$$

In contrast to function $F1$, $F2$ is much more rugged and since the point $(s = 0, t = 0)$ is not Nash, the optimum of $F2$ is not stable. Since the solution candidates aim to minimize, they have a very strong incentive to deviate as soon as the coevolutionary system comes close to $(0,0)$. Because the test cases follow the solution candidates, both leave the optimum quickly (see Fig. 2).

For some visualizations of functions $F1$ and $F2$ see Figures 1 and 2.

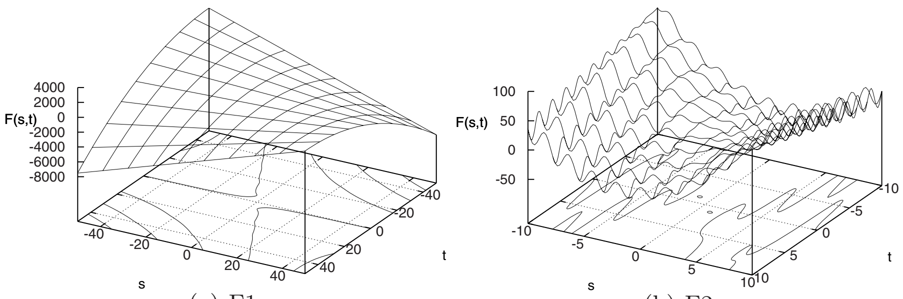

Fig. 1. Visualization of test functions $F1$ and $F2$

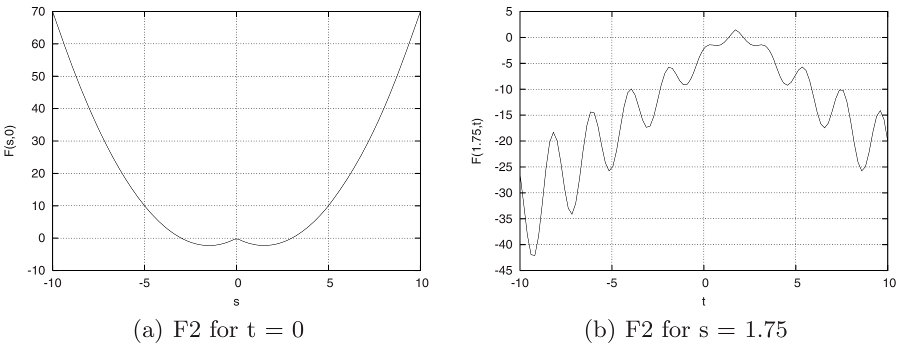

Fig. 2. With Function $F2$, the solution candidates have an incentive to deviate from the optimum (left panel) which as a consequence, causes the test cases to follow (right panel)

4 Fitness Assignment Methods

In this section, we examine various fitness assignment methods, i.e., methods which calculate an individual's fitness based on the results of testing all solutions against all test cases, resulting in $|P_S| \times |P_T|$ function evaluations.

Let us denote the populations before selection as P_S and P_T, and the (smaller) populations after selection as P'_S and P'_T. Then, we distinguish between three different rankings/fitness values. By *real* ranking, we mean a ranking based on the solution's performance with respect to the true worst case (i.e., $t = s$). We call the ranking based on the populations before selection as *global* ranking, and the ranking based on the populations after selection the *local* ranking. W.l.o.g., best solutions have the lowest fitness, while for test cases, a higher fitness is assumed to be better. Note that the local fitness is always at least as good as the global fitness, as additional test cases can only worsen performance.

The solutions are always ranked according to their respective global worst case performance (over all test cases).

$$fit(s) = \max_{t \in P_T} F(s,t)$$

Furthermore, let us assume that solutions are numbered from 1 to n in order of increasing fitness (increasing global worst case values), i.e., the currently best perceived solution in the population is denoted by s_1.

In the following, we first describe two fitness assignment methods from the literature, namely the Maximin method and Jensen's method. Then, we continue to propose some new approaches.

Maximin Method. To rank the test cases, the classical approach is to also use the minimax principle, see, e.g., (2, 6, 8, 15). Since they represent the opposite perspective, the correct term is Maximin method.

$$fit_{Maximin}(t) = \min_{s \in P_S} F(s,t)$$

(8) shows, however, that this approach fails to find the optimum, if the concerned function is not a saddle-point function.

Jensen's Method. This approach is described in detail in (9). Jensen argues that the fitness of a test case should not only rely on the performance of that particular test case on P_S, but also on other test cases in P_T. If at least one solution exists, for which a test case forms a very difficult (or the worst) case, this test case should get a high fitness, even if it is easy to solve for the other solutions. Therefore in Jensen's approach, a test case's fitness equals the highest ranking it achieves if all test cases are ranked for each solution, the worst case having the highest rank. If a test case achieves this highest rank for $k > 1$ solutions, its fitness is additionally increased.

$$fit_{Jensen}(t) = \max_{s \in P_S} rank_s(t) + \frac{k}{|P_S| + 1},$$

where $rank_s(t)$ is the rank of solution s according to test case t.

The following methods are new, and select test cases one by one, in an iterative and greedy manner, trying to maximize the information about the individuals in P_S that the selected test cases provide. These test cases implicitly serve as a kind of memory, and influence the test cases future solutions will face.

Note that because we use a simple $(\mu + \lambda)$ selection strategy, the only relevant decision is which test cases survive to the next generation. However, the methods could be adapted to other selection methods in a straightforward way by assigning them ranks according to the order in which they are selected. Also, it is possible that the criterion to select the next test case is not unique. In this case, one of the test cases is added randomly.

Worst Case Method. The underlying idea of this method is that it is most important to keep the information about the worst cases of the best solutions (as these will be used to generate offspring). The method starts by going through all solutions $s_1 \dots s_n$ in order of increasing fitness, and, in each iteration, adding the corresponding worst case test case if it is not yet included.

Average Greedy Method. The Average Greedy method is based on the assumption that the local performance reflected by the selected test cases should be as close as possible to the global performance according to all test cases in the population. Therefore, the method starts by selecting the worst case of s_1. Then, it iteratively adds as next test case the one which maximizes the average local fitness of all solutions after adding the additional test case. More formally, if B denotes the set of test cases selected so far, it adds the test case $t' \in P_T$ which maximizes $\sum_{s \in P_S} max_{t \in \{B \cup t'\}} F(s,t)$.

Distance Greedy Method. The Distance Greedy method is based on the observation that a solution's fitness can only deteriorate if additional, more difficult test cases are found. To avoid that the best solution is no longer best in the subsequent iteration, it is attempted to maximize the difference in local fitness between the best and the closest competitors. Again, the method starts by selecting the worst case for s_1. Then, it iteratively adds the test case which, if added to the already selected solutions (B), maximizes the local fitness difference between the best solution and the second best (according to the ranking based on B). In case of ties, the difference between best and third best, fourth best, etc. is used as a criterion. Usually, this method leads to a correct local ranking of the best solutions, as the worst cases for these solutions are often selected first.

Ranking Greedy Method. Here, the motivation is to maintain the relative ordering of all solutions in P_S with only the selected solutions, i.e., to make the local ranking as consistent as possible with the global ranking. Again, the worst case of s_1 is always selected. Then, iteratively the test case is added to B that, if added, maximizes the correct number of relative orderings in the ranking specified by B.

Table 1 demonstrates how the different methods rank the test cases. The particular example consists of six solutions and six test cases, and assumes that

Table 1. Left: Evaluation of six solutions by six test cases (smaller numbers are better). Right: Fitness values assigned to test cases by the different methods (higher numbers are better).

	t_1	t_2	t_3	t_4	t_5	t_6
s_1	3	5	8	9	11	10
s_2	7	9	13	2	4	5
s_3	4	6	7	9	3	2
s_4	12	5	8	9	10	11
s_5	5	6	7	**8**	2	3
s_6	2	8	9	10	8	6

	t_1	t_2	t_3	t_4	t_5	t_6
Maximin	2	5	7	**2**	2	2
Jensen	5	4	5	**5.43**	5	4.29
Worst Case	3	1	2	**5**	4	0
Average Greedy	3	1	4	**5**	2	0
Distance Greedy	1	4	2	**5**	3	0
Ranking Greedy	2	1	4	**5**	3	0

in the case of ties, always the test case with the smaller index is chosen. As can be seen, the methods value the test cases quite differently, and it is not obvious which ranking is best. In any case, note that the Maximin method gives a very low evaluation to the test case causing worst case performance of the best solution (t_4, bold). Thus, it is likely that this test case does not survive to the next iteration, which, from our experience, would be very important. Jensen's method gives this test case the highest evaluation in this example, although this is not guaranteed in general. All our newly proposed methods include this test case with highest priority in the next population.

5 Metrics

Various metrics are used to analyze the performance of the different fitness assignment methods. The most important one is the *real fitness* of the generation's perceived best solution. Applying the knowledge that the real worst case is $t = s$, the real fitness can be calculated as $F(s_1, s_1)$.

The *correlation coefficient* between the real fitness ($F(s_1, s_1)$) and the subjective fitness of a solution ($fit(s_1)$) indicates the method's ability to keep real and subjective fitness linked to each other. It's measured across all individuals of the population.

A similar metric we designed is called *real quota*. It measures the fraction of the μ objectively (according to real fitness) best solutions the method succeeds to identify by counting how many of them are among the μ *subjectively* best solutions, i.e., whether the selection of μ parents is correct. A real quota of 1 means a perfect match.

Both metrics, correlation coefficient and real quota serve only for monitoring. The information they use is not accessible to the CEA and thus cannot be used to direct the fitness assignment process. A metric which uses only information that is acquirable for the CEA is the *global quota*. It counts how many of the μ subjective, i.e., global, best solutions are among the μ *local* best solutions. This metric gives information about the fitness assignment process in the test case population, which can be designed to optimize the global quota.

6 Empirical Results - Fitness Assignment

Experimental Setup. In this section, we use a $(10+20)$ evolutionary strategy with Gaussian mutation and step size 0.35. All values are mean values over 400 runs. Unless specified otherwise, the plotted values depict mean value and standard error of the respective metric.

Figure 3 displays the evolution of real fitness of the perceived best solution over time for test function $F1$. As can be seen, all fitness assignment methods except the method proposed by Jensen are able to converge to the optimum on this simple problem. The minimax method converges significantly slower, the newly proposed fitness assignments all perform similar. Additional experiments have shown that Jensen is also able to converge on this problem when allowed a larger population size.

The same plot, but for function $F2$, is shown in Figure 4. Here, the performance differences are much more significant. None of the algorithms is able to converge to the optimum, which was to be expected, since the function rewards deviations from the optimum. Average Greedy and Ranking Greedy perform best, followed by Distance Greedy and Jensen's method. The Worst Case method works very well in the first few iterations, but then suddenly deteriorates and converges to a level much worse than what had been obtained before. The good performance in the first phase can be explained by the uncompromising focus on the worst cases, driving the solution values down. The following ascent may be explained by an overspecialization in P_T. Very few test cases form the complete set of worst cases for all solutions, resulting in less than μ test cases with an assigned fitness. Therefore, some test cases are chosen randomly, substantially worsening the algorithm's performance.

The Maximin approach actually diverges and results in solutions worse than the random initial population.

Global Quota and Correlation Coefficient. The basic assumption behind the greedy approaches was that the performance of the CEA could be improved by maintaining, in the local information, as much of the global information as possible. The success of this idea is reflected in the sound performance of

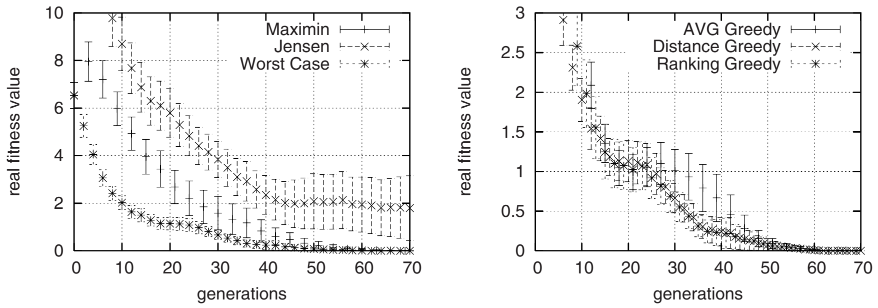

Fig. 3. Real fitness values and standard error for function $F1$ (optimum is 0.0)

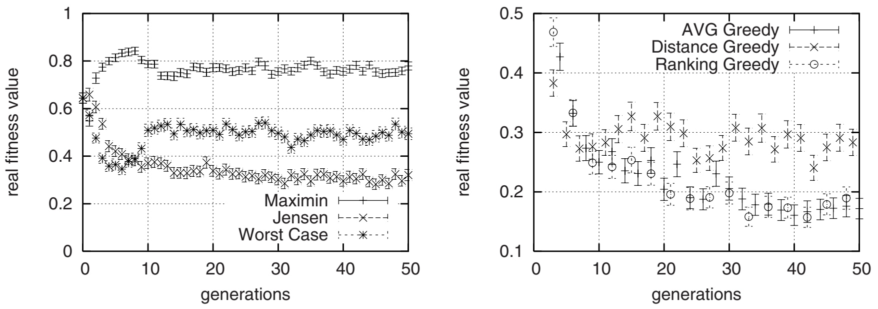

Fig. 4. Real fitness values and standard errors for function $F2$ (optimum is 0.0)

Table 2. Correlation coefficient and global quota ± standard error in generation 100 for function $F2$

Method	Correlation Coefficient	Global Quota
Maximin	0.73 ± 0.017	0.54 ± 0.018
Jensen	0.8 ± 0.016	0.814 ± 0.007
Worst Case	0.71 ± 0.018	0.475 ± 0.018
Average Greedy	0.89 ± 0.01	0.87 ± 0.006
Distance Greedy	0.837 ± 0.013	0.84 ± 0.007
Ranking Greedy	0.9 ± 0.009	0.924 ± 0.006

the greedy methods on both problems. It can also be measured by the global quota and the correlation coefficient as reported in Table 2. The Ranking Greedy method was the last greedy method to be designed and it was especially developed to further improve the global quota, which was clearly successful. Nevertheless the Ranking Greedy method does not outperform the Average Greedy method regarding the real fitness, indicating that the connection between global quota, real quota and the correlation coefficient respectively seems to be more complex than expected. Table 2 shows that Average Greedy and Ranking Greedy have the same correlation coefficient, stating that both achieve about the same correlation between subjective and objective fitness.

7 Empirial Results - Mutation Step Size

The difficulty of function $F2$ lies in the fact that once the test case population converged to the worst case ($t = s$), the optimum is surrounded by a much more attractive area for the solutions. Therefore, they mainly circle around $(0, 0)$, always followed by the test cases. In order to drive the solutions back to $(0, 0)$, the test cases must "overtake" the solution value. This insight led us to test a mutation step size for the test cases larger than the mutation step size for the solutions.

To analyze the relation between the mutation step sizes of the two populations, the mutation step size of the solution population was fixed to a standard

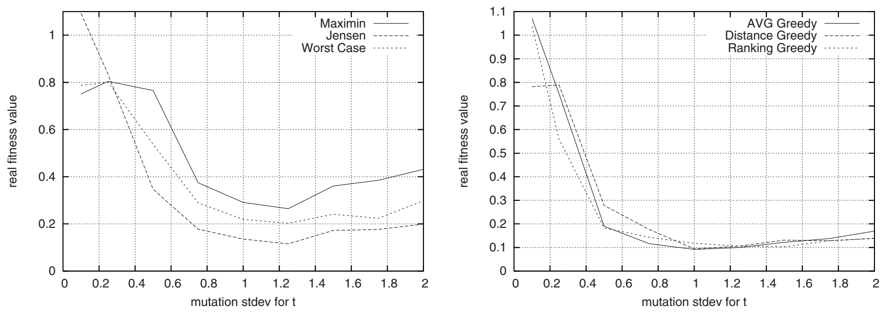

Fig. 5. Function $F2$: Best solution found in Generation 100. Mutation step size for P_S is fixed to 0.5 and varies for P_T.

deviation $\sigma = 0.5$ while various values were tested for the test case population. The real worst case fitness of the last generation's perceived best solution was plotted for each combination. The mean values over 400 runs can be seen in Fig. 5. The standard errors are very small, and have been omitted in the plot for clarity. Best performance is reached if the test cases' mutation step size is a bit more than double of the solutions' step size. So, our initial assumption has been confirmed. Increasing also the solution population step size to the higher value again lead to worse performance (not shown).

8 Conclusion

Coevolutionary algorithms seem an efficient and promising approach to worst-case optimization. In this paper, we have proposed and analyzed a number of variants of coevolutionary algorithms. The focus of our study was on new ways to determine the fitness of the test cases. Here, we proposed several greedy mechanisms which aim at preserving as much worst-case information about the good solutions as possible after selection. As has been shown empirically, the new methods significantly outperform previously proposed fitness assignment schemes on the suggested test functions.

Besides, we have experimented with different mutation rates for the solution and test case populations, and found that it is beneficial to choose a higher mutation rate for the test population than for the solution population.

Overall, this paper has proposed several novel and promising ways to improve the performance of coevolutionary worst-case optimizers. As a next step, the obtained results should be confirmed on a variety of additional test problems. Also, it would be straightforward to use the various methods in a lexicographic order, and switch from one to another in case of ties.

Acknowledgement. We would like to thank Elena Popovici from Icosystem Inc. for interesting discussions.

References

1. Avigad, G., Branke, J.: Worst-case robustness and related decision support. In: Genetic and Evolutionary Computation Conference, ACM Press, New York (to appear)
2. Barbosa, H.J.C.: A coevolutionary genetic algorithm for constrained optimization. In: Congress on Evolutionary Computation, vol. 3, pp. 1605–1611 (1999)
3. Branke, J.: Evolutionary Optimization in Dynamic Environments. Kluwer Academic Publishers, Norwell (2001)
4. Daum, D.A., Deb, K., Branke, J.: Reliability-based optimization for multiple constraints with evolutionary algorithms. In: Congress on Evolutionary Computation, pp. 911–918. IEEE Computer Society Press, Los Alamitos (2007)
5. de Jong, E.: The maxsolve algorithm for coevolution. In: Conference on Genetic and Evolutionary Computation, pp. 483–489. ACM Press, New York (2005)
6. Herrmann, J.W.: A genetic algorithm for minimax optimization problems. In: Congress on Evolutionary Computation, vol. 2, pp. 1099–1103. IEEE Computer Society Press, Los Alamitos (1999)
7. Hillis, D.W.: Co-evolving parasites improve simulated evolution in an optimization procedure. Physica D 42, 228–234 (1990)
8. Jensen, M.T.: Finding worst-case flexible schedules using coevolution. In: Spector, L., et al. (eds.) Genetic and Evolutionary Computation Conference, pp. 1144–1151. Morgan Kaufmann, San Francisco (2001)
9. Jensen, M.T.: A new look at solving minimax problems with coevolutionary genetic algorithms. Applied Optimization 86, 369–384 (2004)
10. Korn, R., Steffensen, M.: On worst-case portfolio optimization. SIAM Journal on Control and Optimization 46(6), 2013–2030 (2007)
11. Luke, S., Wiegand, R.P.: When coevolutionary algorithms exhibit evolutionary dynamics. In: Barry, A.M. (ed.) GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference, pp. 236–241. AAAI Press, Menlo Park (2002)
12. Ong, Y.-S., Nair, P.B., Lum, K.Y.: Max-min surrogate-assisted evolutionary algorithm for robust design. IEEE Transactions on Evolutionary Computation 10(4), 392–404 (2006)
13. Pagie, L., Hogeweg, P.: Information integration and red queen dynamics in coevolutionary optimization. In: Proceedings of the 2000 Congress on Evolutionary Computation, vol. 2, pp. 1260–1267 (2000)
14. Paredis, J.: Coevolutionary computation. Artificial Life 2(4), 355–375 (1995)
15. Sebald, A.V., Schlenzig, J.: Minimax design of neural net controllers for highly uncertain plants. IEEE Transactions on Neural Networks 5(1), 73–82 (1994)