Focusing Search in Multiobjective Evolutionary Optimization through Preference Learning from User Feedback

Thomas Fober, Weiwei Cheng, and Eyke Hüllermeier

Department of Mathematics and Computer Science
University of Marburg, Germany
{thomas,cheng,eyke}@informatik.uni-marburg.de

Abstract

The problem of computing the set of Pareto-optimal solutions in multiobjective optimization has been tackled by means of different approaches in previous years, including evolutionary algorithms. A key advantage of computing the whole set of Pareto-optimal solutions is *completeness*: None of the solutions that might be maximally preferred by the user is lost. An obvious disadvantage, however, is the computational effort, which may become prohibitive for high-dimensional problems. Besides, the resulting set itself may become rather large, making it impracticable to present the entire set to the user. In order to mitigate these problems, we propose a method for incorporating user-feedback in the optimization process by asking the user for *pairwise preferences*. The pairwise preferences thus produced are then used as training examples for a preference learning method leading to a hypothetical utility model that approximate the true but unknown (latent) utility function of the user. This model is used to focus the search on the most promising parts of the Pareto front, which is approximated by an evolutionary algorithm.

1 Introduction

In multiple criteria (aka multiobjective) optimization, the goal is to simultaneously optimize a set of potentially conflicting criteria specified in terms of a set of objective functions. A problem of this kind is often transformed into a single-criterion problem first, for example by replacing the original criteria by their weighted sum, and then solved using classical optimization techniques afterward. However, since the true weights specifying the importance of each criterion are normally not known, a solution thus produced may not be optimal from the user's point of view.

One possibility to avoid this problem is to compute the complete set of Pareto optimal solutions, thereby ensuring that none of the solutions that might be maximally preferred by the user gets lost. The disadvantage of this alternative is the computational effort it involves, which may become prohibitive for high-dimensional problems. Besides, the Pareto front itself may become rather large, making it impracticable to present the entire set to the user and expecting her to choose the most preferred solution among these candidates. Finally, noting that this set is necessarily a finite approximation of the Pareto front, a solution sufficiently close to a truly optimal one is not guaranteed, so there is still a danger of ending up with a suboptimal solution.

The two approaches above are in a sense extreme opposites: While the first one pretends perfect knowledge of the user preferences from the very beginning, the second one remains in this regard completely agnostic till the very end. The idea of this paper is to find a trade-off between both extremes. To this end, we propose a method for incorporating user-feedback in the optimization process. This feedback is used in order to focus the search on the most promising parts of the Pareto front, which is approximated by a modification of the NSGA-II algorithm.

More specifically, given the current approximation of the set of Pareto-optimal solutions, the idea is to ask the user for *pairwise preferences*: Two solutions from this set are selected, and the user is asked which of these solutions is preferred. The pairwise preferences thus produced are then used as training examples for a preference learning method, that is, a machine learning method for inducing a (hypothetical) utility model that approximates the true but unknown (latent) utility function of the user. This model, in turn, is then used for restricting the current set of Pareto-optimal solutions to those candidates with the highest (estimated) utility, and this part of the Pareto front is explored in more detail. The whole process iterates until a termination condition is met.

Our learning algorithm is inspired by methods that have recently been developed in the fields of preference learning and learning to rank. It is implemented as an online perceptron learner that exhibits a number of interesting properties. Most notably, it learns in an *incremental* way, which means that new user feedback can be incorporated efficiently without losing the feedback from previous iterations. Moreover, it is robust toward noise and guarantees the learned utility function to be monotone increasing in all criteria. Finally, it also supports an *active learning* strategy, in which pairwise preference judgments are not requested for solutions drawn from the Pareto front at random, but instead for pairs of solutions that are supposedly most informative from a learning point of view.

The remainder of the paper is organized as follows. Subsequent to a brief introduction to the problem of multiobjective optimization in Section 2, our novel approach to enhancing corresponding solvers by means of machine learning methods is introduced in Section 3. Experimental results are presented in Section 4. Finally, Section 5 concludes the paper.

2 Multiobjective Optimization

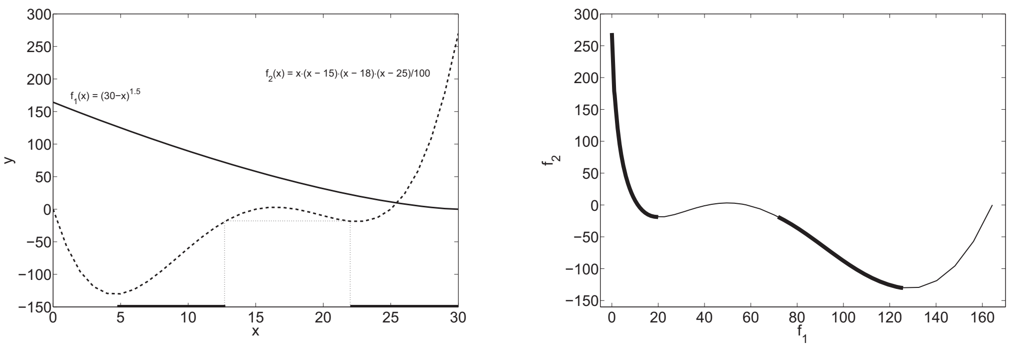
In multiobjective optimization, a set of functions f_i, $i = 1, \dots, m$ is given that have to be optimized simultaneously. In this paper, we will focus on real-valued optimization problems, hence the functions are of the form $f_i : \mathcal{S} \subseteq \mathbb{R}^n \to \mathbb{R}$. Moreover, and without loss of generality, we consider minimization problems of the form

$$\begin{aligned} \text{minimize } &\{f_1(x), \dots, f_m(x)\} \\ \text{subject to } &x \in \mathcal{S} \subseteq \mathbb{R}^n \end{aligned} \quad . \tag{1}$$

To define optimality, we use a partial ordering $\succ_p$ on $\mathbb{R}^m$. For two vectors $v, v' \in \mathbb{R}^m$, we write $v \succ_p v'$ if and only if $[v]_i \leq [v']_i$ for all $i = 1, \dots, m$ and $[v]_i < [v']_i$ for some $i = 1, \dots, m$. This ordering can be used to define Pareto optimality for the multiobjective problem: A solution $x \in \mathcal{S} \subseteq \mathbb{R}^n$ is Pareto optimal if there exist no $x' \in \mathcal{S} \subseteq \mathbb{R}^n$ such that $\boldsymbol{f}(x) \succ_p \boldsymbol{f}(x')$, where $\boldsymbol{f}(x) = (f_1(x), \dots, f_m(x))^T$. The solution of (1) is then given by the set of all Pareto optimal solutions.

In general, since the objectives might be conflicting, there will be more than one Pareto-optimal solution. Figure 1(a) shows an example for a two-criteria optimization problem with two conflicting criteria: With increasing x, the function f_1 is decreasing whereas

f_2 is increasing. Having the goal to minimize both functions, the optimal solutions in a Pareto sense will be those indicated by the blue bars on the x-axis.



(a) Decision space; Pareto-optimal solutions are marked by a blue bar on the x-axis

(b) Objective space defined as $f_1 \longrightarrow f_2$ mapping; Pareto optimal set is marked by a blue bold line

Figure 1: Minimization of the function $f : [0, 30] \to \mathbb{R}^2$ defined by $x \mapsto (f_1(x), f_2(x))^T$, where $f_1(x) = \sqrt{(30 - x)^3}$ and $f_2(x) = x(x - 15)(x - 18)(x - 25)/100$

Instead of considering the decision space $\mathcal{S} \subseteq \mathbb{R}^n$, one can directly consider the image of the decision space, namely the objective space that consists of vectors $\boldsymbol{f}(x) = (f_1(x), \dots, f_m(x))^T \in \mathbb{R}^m$. This space is illustrated in Figure 1 (b) for the example of two objective functions. Here, the optimal Pareto front is visualized in terms of a bold line.

2.1 Principles to Solve Multiobjective Optimization Problems

As already mentioned in the introduction, multiobjective optimization problems can be tackled in different ways. Frequently used principles include methods transforming the multiobjective problem into a single-criterion problem, methods which try to approximate the complete Pareto front, and interactive approaches that interact with the user during optimization process.

A transformation of a multiobjective optimization problem into a single-criterion problem can be accomplished in different ways. The simplest one is to specify a weight for each function and to use the weighted sum as an objective function. Another idea is to specify a hypothetical optimal (though not necessarily feasible) multiobjective solution the user is aiming for and to minimize the distance to this preferred solution. It is also possible to start with a single criterion and to optimize it. Based on the solution obtained, bounds are specified for the remaining criteria, thus defining $(m - 1)$ additional constraints. Independent of the concrete technique chosen, a single-criterion function is obtained that can be optimized with state-of-the-art techniques.

The above reduction schemes are clearly appealing at first sight, especially from a computational point of view. One should note, however, that single-objective optimization is not necessarily an easy problem. Another problem is that weights, hypothetical points or bounds have to be set beforehand, while information gathered in the course of the optimization process is not explicitly used. Finally, different functions can have different ranges and/or scales, which makes their combination difficult.

To overcome these problems, one can approximate the complete Pareto front and present it to the user who can finally select an element she prefers most. For example, [6, 10] use evolutionary algorithms which have the benefit of considering a population able to approximate the Pareto front. The Pareto front, however, can become very large, especially in high dimensions, leading to computational problems and complexity issues. For the user, it might be difficult to select a solution from a high-dimensional Pareto front. In fact, visualization of Pareto sets in higher dimensions is an intricate problem for which different approaches have been proposed, such as scatter-plot matrices, Chernoff faces, parallel coordinates or animated meshes.

The third principle focuses on an interactive process with "the user in the loop". Roughly speaking, the idea is to exploit user feedback for guiding the optimization process in one way or the other, notably by learning a kind of utility model. This model can then be used to focus the search or to transform the problem into a single-criterion one. Methods used for this purpose include ordinal regression [3], rough sets [7] as well as similarity-based approaches [11].

3 Interactive Multiobjective Evolutionary Optimization

In evolutionary optimization, the NSGA-II algorithm [6] is the standard approach to solving multiobjective optimization problems. This algorithm essentially consists of a loop which is executed after initialization of a start population. By applying mating-selection, recombination and mutation, a new set of individuals is generated which is used (together with the current population) to select individuals for the next generation. Since we consider real-valued optimization problems, we take the operators mating-selection, recombination and mutation from evolutionary strategies [2].

The selection operator of the NSGA-II algorithm developed for multiobjective problems tries to approximate the complete Pareto set. This is done by sorting the population according to dominance, leading to subpopulations $\mathcal{F}_1$ containing all individuals which are non-dominated, $\mathcal{F}_2$ containing those individuals dominated by $\mathcal{F}_1$, and so on. Moreover, to distribute individuals uniformly on the Pareto front, a "crowding-distance" is used which leads to a preference for individuals with no or few other individuals in their neighborhood. Selection is now performed by successively taking the sets $\mathcal{F}_i$ $(i = 1, 2, \dots)$ into the next population until there is no space left for a certain $\mathcal{F}_j$. To fill the complete population, the individuals in $\mathcal{F}_j$ are sorted according to their crowding-distance. Jointly, these two mechanisms lead to an approximation of the complete optimal Pareto front.

In this paper, we propose a modification of the (environment) selection operator, since our goal is not to approximate the Pareto front in a uniform way. On the contrary, our goal is to compute a subset of the Pareto front which is maximally in accordance with the preferences of the user.

3.1 Focused Evolutionary Strategy

To reach our goal of focusing the search to a region of solutions the user prefers, we substitute the crowding distance by a utility model that seeks to approximate the (latent) utility function of the user. The modification of the NSGA-II algorithm thus obtained will be called *Focused Evolutionary Strategy (FES)*. For the time being, suppose we have a technique to calculate a utility model from a set of preferences "$\boldsymbol{f}(x)$ is preferred to $\boldsymbol{f}(x')$", also written $\boldsymbol{f}(x) \succ_u \boldsymbol{f}(x')$. A concrete technique of that kind will be introduced in Section 3.3.

Our algorithm starts as usual with an initialization of the population. After initialization, however, first the utility model must be trained, before the evolutionary loop can be entered. Having determined the utility model as described in Section 3.3, the evolutionary loop is executed, which, as already mentioned, is adopted from evolutionary strategies. Since the corresponding selection operator cannot be applied here, we use non-dominated sorting as in NSGA-II. However, since our goal is to compute a subset of the Pareto front which is maximally preferred by the user, instead of approximating it uniformly, we replace the crowding distance by our utility model for selecting individuals.

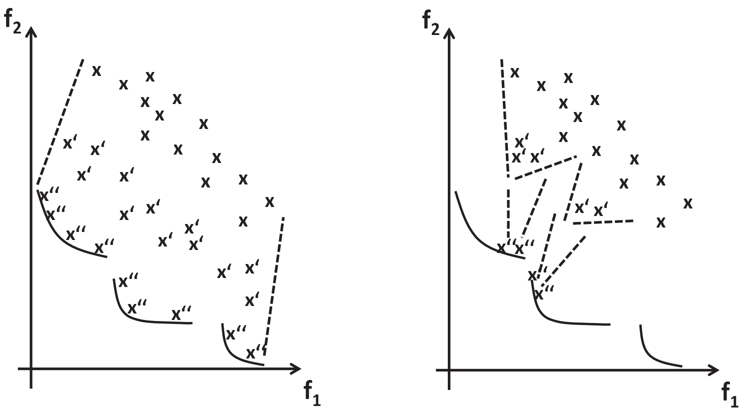


Figure 2: Difference between the NSGA-II approach (left) and the FES approach (right). While the former seeks to approximate the Pareto front in a uniform way, the latter tries to focus on solutions that are presumably preferred by the user.

An illustration is given in Figure 2. In the left picture, the complete Pareto optimal set is approximated. Consequently, the intermediate solutions (the progress in terms of generations is indicated by the number of primes) are spread over a large interval also containing solutions that are probably not of high interest for the user. In the right picture, the search is focused to the high-utility solutions. Hence, the number of solutions the algorithm has to consider is much smaller, and the algorithm therefore faster. Moreover, for the user, it is arguably much more convenient to select the final solution from a reduced set of candidates, all of which are probably close to the ideal solution (i.e., the solution having the highest utility for the user).

3.2 Focused Pareto Selection

The selection operator *FPS* must favor solutions that are closer to the optimal Pareto front and solutions that are preferred by the user, i.e., which have a high value according to the (latent) utility function of the user—recall that we merely assume the existence of this function, not that we know it; instead, we seek to learn it.

The FPS operator takes as input the set of individuals used for selection (usually offsprings or parents and offsprings) and performs non-dominated sorting on this set. This step is hence equal to NSGA-II. Let $\mathcal{F}$ denote the union $\mathcal{F}_1 \cup \mathcal{F}_2 \cup \ldots \cup \mathcal{F}_j$, where j is the smallest index assuring that $|\mathcal{F}| \geq \mu$. Thus, $\mathcal{F}$ contains enough individuals to fill the next population.

The individuals in $\mathcal{F}$ are then sorted according to the current utility model, and the μ best ones are selected. Our utility model is derived from an ensemble of real-valued utility functions u_i, $i = 1, \ldots, k$. As will become clear later on, the diversity of this ensemble reflects the uncertainty about the true utility function u^* we seek to approximate. The utility assigned to an individual x is then defined as

$$u(x) = \max_{1 \leq i \leq k} u_i(x) \enspace .$$

This optimistic evaluation is obviously in agreement with our goal of not losing any promising solution. Our selection procedure is illustrated in Figure 3.

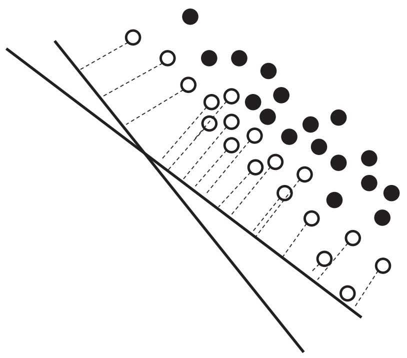


Figure 3: Illustration of the FPS selection mechanism: First, candidate solutions are selected according to the dominance criterion (non-filled balls). These solutions are then evaluated by the utility model. Here, the dashed lines indicate the assigned utility values (the shorter the line, the larger the utility).

3.3 Learning Preferences and the Utility Model

In the following, we present more details about our utility model. Basically, we assume linear utility functions of the form

$$u(x) = u(\boldsymbol{f}(x)) = -\langle \boldsymbol{w}, \boldsymbol{f}(x) \rangle \; = \; -\sum_{i=1}^{m} w_i \cdot f_i(x) \; , \tag{2}$$

where $\boldsymbol{w} = (w_1, \ldots, w_m)^T$ is a weight vector and $\boldsymbol{f}(x)$ the vector of the m objective functions. Each such function induces a total order on the objective space: Given $\boldsymbol{f}(x) = (f_1(x), \ldots, f_m(x))$ and $\boldsymbol{f}(x')$,

$$\boldsymbol{f}(x) \succ_u \boldsymbol{f}(x') \; \overset{\text{df}}{\Longleftrightarrow} \; u(\boldsymbol{f}(x)) \geq u(\boldsymbol{f}(x')).$$

Note that it makes sense to require

$$\boldsymbol{w} = (w_1, \ldots, w_m)^T \; \geq \; 0 \tag{3}$$

in order to guarantee the monotonicity of the function (2). That is, if one of the criteria f_i increases, while all others remain unchanged, then the utility can only decrease. In this case, it is also guaranteed that the optimum of (2) is a Pareto-optimal solution of (1).

A linear model is quite attractive from a machine learning point of view, as it is amenable to efficient learning algorithms and, moreover, to non-linear extensions via "kernelization" [12]. The learning problem itself comes down to estimating the weights w_i in (2). To this end, the user is asked for training information in the form of *qualitative* feedback, which is typically much easier to acquire than absolute feedback. More specifically, the user is expected to provide pairwise comparisons of the form '$\boldsymbol{f}(x)$ is preferred to $\boldsymbol{f}(x')$'.

A set of c pairs $\boldsymbol{f}(x), \boldsymbol{f}(x')$, each of which is evaluated by the user according to her preferences, is generated as training information. The parameter c allows one to adapt the size of the training set to the application at hand. Since it makes no sense to request the user's preference for a pair of solutions that can be ordered according to $\succ_p$, only those pairs are considered that belong to the same set $\mathcal{F}_i$ of incomparable solutions. First, $\mathcal{F}_1$ is used, followed by $\mathcal{F}_2$ and so forth, until the required training set $\mathbb{T}$ is obtained.

The goal of the learning process is to find a utility function of the form (2) which is as much as possible in agreement with the training set and also satisfies the monotonicity property (3). Besides, this function should of course generalize as well as possible beyond these examples. A key idea in our approach is to reduce the above learning problem to a *binary classification problem*. Due to the assumption of a linear model (2) this is indeed possible: The constraint $u(\boldsymbol{f}(x)) > u(\boldsymbol{f}(x'))$ induced by a preference $(\boldsymbol{f}(x) \succ_u \boldsymbol{f}(x'))$ is equivalent to $\langle \boldsymbol{w}, \boldsymbol{f}(x) - \boldsymbol{f}(x') \rangle > 0$ and $\langle \boldsymbol{w}, \boldsymbol{f}(x') - \boldsymbol{f}(x) \rangle < 0$. From a classification point of view, $\boldsymbol{f}(x) - \boldsymbol{f}(x')$ can hence be seen as a positive example and $\boldsymbol{f}(x') - \boldsymbol{f}(x)$ a negative one. To learn (2), the noise-tolerant perceptron algorithm proposed in [9] is used, leading to a weight vector $\boldsymbol{w} = (w_1, \ldots, w_m)$ eventually defining (2).

3.3.1 Monotonicity

The monotonicity constraint (3) constitutes an interesting challenge from a machine learning point of view. In fact, this relatively simple property is not guaranteed by many standard machine learning algorithms. That is, a model that implements a distance function $u(\cdot)$ may easily violate the monotonicity property, even if this condition is satisfied by all examples used as training data.

Fortunately, our learning algorithm allows us to incorporate the monotonicity constraint in a relatively simple way. The well-known perceptron algorithm is an error-driven on-line algorithm that adapts the weight vector $\boldsymbol{w}$ in an incremental way. To guarantee monotonicity, we simply modify this algorithm as follows: Each time an adaptation of $\boldsymbol{w}$ produces a negative component $w_i < 0$, this component is set to 0. Roughly speaking, the original adaptation is replaced by a "thresholded" adaptation.

In its basic form, the perceptron algorithm provably converges after a finite number of iterations, provided the data is linearly separable. We note that this property is preserved by our modification [5].

3.3.2 Ensembles

Instead of training a single linear utility function (weight vector), we use standard bagging [4] to train an ensemble consisting of k models (weight vectors) $\boldsymbol{w}^{(i)}, i = 1, \ldots, k$, where k is another exogenous parameter. The purpose of this approach is twofold.

First, noting that our current model is only an estimation of the true utility function of the user, we seek to capture the uncertainty of this estimation. This uncertainty is in direct correspondence with the size of the training data set: The larger this set, the more similar the bagging samples and, therefore, the more similar the models $\boldsymbol{w}^{(i)}, i = 1, \ldots, k$ become. Thus, the diversity of the ensemble will decrease in the course of time, by gathering more and more feedback from the user, which clearly makes sense. Indeed, as can be seen in Figure 3, the less diverse the ensemble, the more one will focus on a small subregion of the Pareto front.

Second, as will be detailed next, the ensemble technique is also useful in connection with the selection of informative queries to be given to the user and used to improve the models.

3.3.3 Updating the Utility Function

The probability of updating the utility functions in a certain generation is controlled by a parameter p_{update}. The update process itself requires the generation of new queries, which can simply be done by means of a random selection (i.e., using a uniform distribution on the set of non-dominated tuples from the current population). Indeed, in the first generation of our evolutionary algorithm, this is arguably the most reasonable approach. In subsequent generations, however, queries can be generated in a more sophisticated way.

More specifically, the idea is to select a maximally informative query, i.e., an example that helps to improve the estimation of utility functions as much as possible. This idea of generating maximally useful examples in a targeted way is the core of *active learning* strategies [13]. In the literature, numerous techniques for active learning have been proposed, most of them being heuristic approximations to theoretically justified (though computationally or practically infeasible) methods. Here, we resort to the *Query by Committee* approach [13]. Given an ensemble of utility functions, the idea is to find a pair of functions and a query for which the disagreement between the predictions of these utility functions is maximal. Intuitively, a query of that kind corresponds to a "critical" and, therefore, potentially informative example.

In our case, the utility functions are given by the ensemble of perceptrons. Moreover, given two objective vectors $\boldsymbol{f}(x)$ and $\boldsymbol{f}(x')$, two utility functions u_i and u_j disagree with each other if $u(\boldsymbol{f}(x)) > u(\boldsymbol{f}(x'))$ while $u(\boldsymbol{f}(x)) < u(\boldsymbol{f}(x'))$. Various strategies are conceivable for finding a maximally critical query, i.e., a query for which there is a high disagreement among the ensemble. In our case, however, we are interested in improving all models, hence, for each model an informative example is required. Our current implementation uses the following approach: Let $W = \left\{\boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(m)}\right\}$ be the set of k weight vectors of the k utility functions that constitute the current ensemble. First we are identifying the maximally conflicting pair $(\boldsymbol{w}^{(i)}, \boldsymbol{w}^{(j)})$, i.e., the pair that maximizes $\|\boldsymbol{w}^{(i)} - \boldsymbol{w}^{(j)}\|$. Subsequently, using these two weight vectors, the current population is considered and the pair $(\boldsymbol{f}(x), \boldsymbol{f}(x'))$ maximizing $|(u_i(\boldsymbol{f}(x)) - u_i(\boldsymbol{f}(x'))) - (u_j(\boldsymbol{f}(x)) - u_j(\boldsymbol{f}(x')))|$ is chosen. Eventually, a new query is obtained which must be evaluated by the user and which is finally used to update the utility functions u_i and u_j in an incremental way.

4 Experimental Study

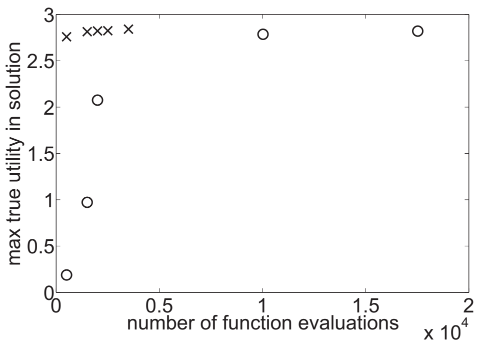
In the experimental study, we consider an artificial set of multiobjective optimization problems and compare our approach with the NSGA-II algorithm, using the same genetic operators in both cases (except the operator used for environment selection). The benchmark set was originally proposed in [8] and consists of 13 multiobjective problems. We chose the functions `OAK2`, `SYM-PART`, `S_ZDT1` and `S_DTLZ2` for our study, since they exhibit different properties like different dimensionality of decision- and objective space, separability, modality and geometry. Moreover, a function $\bar{u}$ is defined to simulate the true user preferences. This is done by using a weighted sum of the criteria, where the weights are drawn randomly in the unit-interval.

We compared NSGA-II and our novel approach FES in terms of the number of function evaluations and the *true* utility value eventually obtained (i.e., the true utility of the best solution in the final population). Obviously, both criteria are related, since the optimization result strongly depends on the number of allowed function evaluations. Here, we consider different numbers of function evaluations, where for a fixed number the exogenous parameters, μ and ν are modified in a systematic way. Moreover, in the case of FES, the parameters c, k and p_{update} are treated in the same manner. Hence, for each value of allowed function evaluations, a set of utility values is obtained, and these two values can be plotted as points in a two-dimensional space.

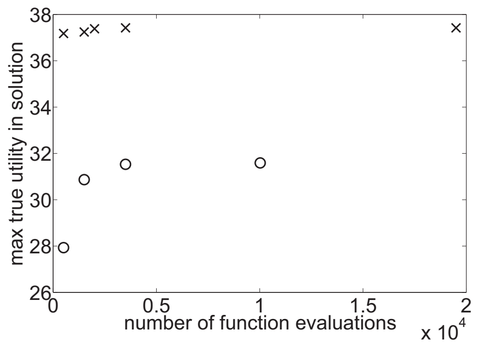
Considering the results illustrated in Figure 4, it clearly turns out that our method compared quite favorably. There are several reasons for this improvement. First of all, NSGA-II is consuming more function evaluations due to the need for approximating the complete Pareto front. Moreover, having found the Pareto front, many individuals are needed for approximating it with a sufficiently high resolution, so as to guarantee a good approximation to the truly optimal solution. The higher the resolution (due to a larger population size), the better the approximation will be. At the same time, however, the complexity will increase, too. In this regard, our approach has a clear advantage, since only a subset of the Pareto front is approximated, which can be accomplished with a smaller population size. Moreover, the utility functions are focusing the search, so that the number of function evaluations which is needed for optimization is further reduced.

5 Conclusions and Outlook

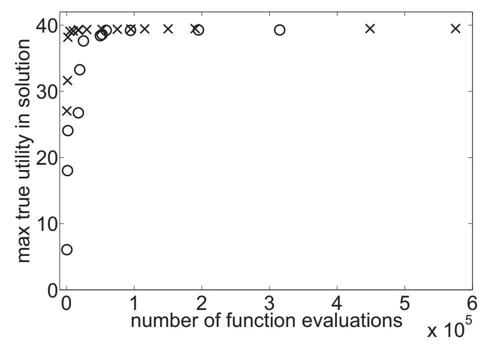
We have proposed a method that employs machine learning techniques for incorporating user-feedback in multiobjective optimization. More specifically, the user is asked to express preferences in the form of pairwise comparisons between candidate solutions. These preferences are used as training examples for a preference learning method leading to a utility model that approximates the true but unknown (latent) utility function of the user. This model in turn is used to focus the search on the most promising parts of the set of Pareto-optimal solutions, which is approximated by an evolutionary algorithm. First



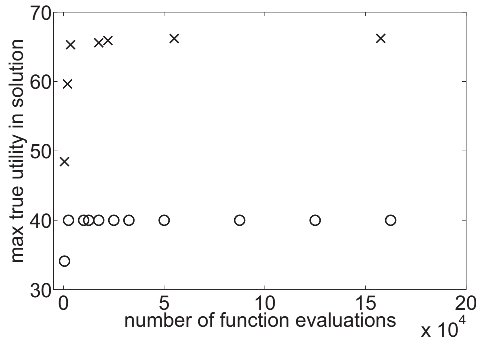
(a) OAK2 $(n = 3, m = 2)$



(b) SYM-PART $(n = 10, m = 2)$



(c) S_ZDT1 $(n = 10, m = 2)$



(d) S_DTLZ2 $(n = 10, m = 8)$

Figure 4: Results on the four benchmark function: Circles and crosses represent, respectively, solutions found by NSGA-II and FES; results of a method which are dominated are removed.

experimental studies using NSGA-II as a benchmark have shown that our approach is able to achieve significant gains in terms of the complexity/utility tradeoff.

As mentioned before, our approach principally allows for substituting the linear model by any other utility model. In this regard, kernels function are of special interest, since they allow to model non-linear dependencies between the objective-functions. In further experiments, more complex utility functions, different from the weighted sum, should therefore be considered. Although we selected a representative subset of the CEC'07 benchmark set, we intent to consider the complete benchmark set together with a more comprehensive parameter study, e.g., by using specialized toolboxes like [1]. Moreover, additional experiments are planned which simulate errors done by the user. Since our learning technique is error-tolerant, we expect a substantial gain in performance even in this case.

References

[1] Bartz-Beielstein, T., Flasch, O., Koch, P., Konen, W.: SPOT: A toolbox for interactive and automatic tuning of search heuristics and simulation models in the R environment. Technical Report, Cologne University of Applied Sciences, Germany (2010).

[2] Beyer, H.-G., Schwefel, H.-P.: Evolution strategies: A comprehensive introduction. Natural Computing, **1** (2002) 3-52

[3] Branke, J., Greco, S., Slowinski, R., Zielniewicz, P.: Interactive evolutionary multiobjective optimization using robust ordinal regression. In: Evolutionary Multi-Criterion Optimization. Springer, Heidelberg, Germany (2009) 554-568

[4] Breiman, L.: Bagging predictors. Machine Learning, **26** (1996) 123-140.

[5] Cheng, W.: Interactive ranking of skylines using machine learning techniques. Master's Thesis, OvG-Universität Magdeburg, Germany (2007).

[6] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, **6**(2) (2002) 182-197

[7] Hernandez-Diaz, A. G., Santana-Quintero, L. V., Coello Coello, C., Caballero, R., Molina, J.: A new proposal for multi-objective optimization using differential evolution and rough sets theory. Proceedings of the 8th annual conference on Genetic and evolutionary computation, Seattle, WA, USA (2006) 675-682

[8] Huang, V. L., Qin, A. K., Deb, K., Zitzler, E., Suganthan, P. N., Liang, J. J., Preuss, M., Huband, S.: Problem definitions for performance assessment on multi-objective optimization algorithms. Technical Report, Nanyang Technological University, Singapore (2007)

[9] Khardon, R., Wachman, G.: Noise tolerant variants of the perceptron algorithm. The Journal of Machine Learning Research **8** (2007) 227–248

[10] Knowles, J.D., Corne, D.W.: Approximating the nondominated front using the Pareto Archived Evolution Strategy. Evolutionary Computation, **8**(2) (2000) 149-172

[11] Krettek, J., Braun, J., Hoffmann, F., Bertram, T.: Preference modeling and model management for interactive multi-objective evolutionary optimization. In: Proc. IPMU 2010, Dortmund, Germany (2010) 584 ff.

[12] Schölkopf, B., Smola, A.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press (2001)

[13] Seung, H., Opper, M., Sompolinsky, H.: Query by committee. In: Computational Learning Theory. (1992) 287–294