# An Adaptive Scheme to Generate the Pareto Front Based on the Epsilon-Constraint Method

Marco Laumanns[1], Lothar Thiele[2], and Eckart Zitzler[2]

[1] ETH Zurich, Institute for Operations Research, CH-8092 Zurich, Switzerland, `laumanns@ifor.math.ethz.ch`

[2] ETH Zurich, Computer Engineering and Networks Laboratory, CH-8092 Zurich, Switzerland, {`thiele,zitzler`}`@tik.ee.ethz.ch`

**Abstract.** This paper presents a scheme for generating the Pareto front of multiobjective optimization problems by solving a sequence of constrained single-objective problems. Since the necessity of determining the constraint value a priori can be a serious drawback of the original epsilon-constraint method, our scheme generates appropriate constraint values adaptively during the run. A simple example problem is presented where the running time (measured by the number of constrained single-objective sub-problems to be solved) of the original epsilon-constraint method is exponential in the problem size (number of decision variables), although the size of the Pareto set grows only linearly. For our method we show that — independent of the problem or the problem size — the time complexity is $O(k^{m-1})$, where $k$ is the number of Pareto-optimal solutions to be found and $m$ the number of objectives. Using the algorithm together with a standard ILP solver for the constrained single-objective problems, the exact Pareto front is generated for the three-objective 0/1 knapsack problem with up to 100 decision variables. Links to problem instances and a reference implementation of the algorithm are provided.

**Keywords.** Multiobjective optimization, generating methods, epsilon-constraint method, knapsack problem

## 1 Introduction

An important task in multiobjective optimization is to find Pareto-optimal solutions. Their knowledge allows a decision maker to learn more about the trade-offs among the different objectives. From both a practical as well as a theoretical viewpoint it is desirable to have a method that is in principle able to generate *all* Pareto-optimal objective vectors. This maximizes the trade-off information for a decision maker in search of a final solution. In addition, the growing interest in approximate methods creates a need for benchmarking, which is often difficult without the knowledge of the true Pareto set.

Whenever a multiobjective problem has a finite number of Pareto-optimal objective vectors, one would expect being able to identify all of them using one of the traditional generating methods proposed in the literature, detailed overviews and discussion of such methods can be found in [1–3]. Many multiobjective optimization algorithms — heuristics as well as exact methods — use variations of these generation methods as a frame, and thereby implicitly follow this assumption.

Most generating methods work by transforming the multiobjective problem into a sequence of parameterized single-objective problems such that the optimum of each single-objective problem corresponds to a Pareto-optimal solution. Thereby, these methods rely on the availability of a suitable single-objective optimization algorithm. In this sense, the generating methods represent metastrategies whose tasks are (i) to determine an appropriate scalarization and (ii) to provide a scheme to vary the parameters. As to the scalarization techniques, a lot is known with respect to the properties of the obtained single-objective optima, e.g., whether they are supported, weakly, or proper Pareto-optimal. However, not much work has been done with respect to the time complexity of the schemes to vary the parameters. Until recently, no scheme has been available that can determine the whole Pareto front by a number of single-objective subproblems which depends only on the cardinality of the Pareto front and not on additional properties such as the location of Pareto-optimal solutions in objective space. In [4] we proposed such a scheme based on the epsilon-constraint method. We proved the correctness of the new algorithm and that its running time, measured by the number of calls of a single-objective optimizer, is bounded by $O(k^{m-1})$, where $k$ is the number of Pareto-optimal objective vectors and $m$ the number of objectives. In contrast, the running time of the original epsilon-constraint method [5] is determined by the product of the ratio of the range to the minimum distance between two solutions in each objective. This expression is at least of order $k^{m-1}$, but can also be exponential in $k$ as was shown on a simple two-objective example. A further advantage besides the considerably lower worst-case time complexity of the new method is that it alleviates the necessity to guess an appropriate grid size because the constraint values are adaptively modified during the run.

The purpose of this paper is to briefly recap the ideas and results from [4] and to present an alternative algorithm. Instead of using box constraints (upper and lower bounds), the new algorithm uses only lower bounds, i.e., less constraints in the single-objective subproblems. Empirical tests on the knapsack problem have shown that this improves the solution time of the underlying ILP solver considerably. In addition, the problem of dealing with weakly Pareto-optimal solutions has been solved more elegantly by two subsequent calls of the single-objective optimization algorithm. The new algorithm is presented visually in a flow chart, which should make reimplementation even easier.

## 2 Problem Scenario

The problem we are dealing with is to find the Pareto front of a general multi-objective problem with $m$ objectives. We assume that all objective functions are to be maximized. We assume that the Pareto front is finite.

**Definition 1 (Pareto optimality).** *Let* $\mathbf{f} : X \to F$ *where* $X$ *is called* decision space *and* $F \subseteq \mathbb{R}^m$ objective space*. The elements of* $X$ *are called* decision vectors *and the elements of* $F$ objective vectors*. A decision vector* $\mathbf{x}^* \in X$ *is* Pareto optimal *if there is no other* $\mathbf{x} \in X$ *that* dominates $\mathbf{x}^*$*, where* $\mathbf{x}$ *dominates* $\mathbf{x}^*$*, denoted as* $\mathbf{x} \succ \mathbf{x}^*$*, if* $f_i(\mathbf{x}) \geq f_i(\mathbf{x}^*)$ *for all* $i = 1, \ldots, m$ *and* $f_i(\mathbf{x}) > f_i(\mathbf{x}^*)$ *for at least one index* $i$*. The set of all Pareto-optimal decision vectors* $X^*$ *is called* Pareto set*.* $F^* = \mathbf{f}(X^*)$ *is the set of all Pareto-optimal objective vectors and denoted as* Pareto front*.*

In cases where several Pareto-optimal decision vectors map to the same Pareto-optimal objective vector, we are satisfied with having one representative decision vector for each Pareto-optimal objective vector. This corresponds to finding one optimal solution in the single-objective case [3].

Several methods exist devoted to this task, usually referred to as "a posteriori" or "non-dominated solution generation" methods [1]. They typically define a set of differently parameterized single-objective surrogate problems and apply multiple runs of a single-objective optimizer. The choice of the parameter values determines, which specific elements of the Pareto set are found. It is in general a difficult and sometimes impossible task to choose a sequence of parameter values such that the *whole* Pareto front is discovered. Consider for example the popular methods based on the aggregation of the different objectives via a weighted sum. Different weight vectors would ideally lead to finding different elements of the Pareto front, but for an unknown problem it is not clear what weight combination to choose. Even if all possible weight combinations were used, it cannot be guaranteed to find Pareto-optimal solutions in concave regions of the Pareto front.

Another traditional method from the field of multiobjective optimization to generate the whole Pareto front is the epsilon-constraint method [5]. The epsilon-constraint method works by choosing one objective function as the only objective and the remaining objective functions as constraints. By a systematic variation of the constraint bounds, different elements of the Pareto front can be obtained. The method relies on the availability of a procedure to solve constrained single-objective problems. We abstract from the details of this procedure and simply assume that it terminates after a fixed time $T$ and returns either the optimum of the constrained single-objective problem (cSOP)

$$\begin{array}{ll} \text{maximize} & \Phi(\mathbf{f}(\mathbf{x})) \equiv \Phi(f_1(\mathbf{x}, \ldots, f_m(\mathbf{x})) \\ \text{subject to} & f_i(\mathbf{x}) > \epsilon_i \quad \forall i \in \{1, \ldots, m\}, \\ & \mathbf{x} \in X, \end{array} \tag{1}$$

where $\mathbf{f} : X \longrightarrow \mathbb{R}^m$ and $\Phi : \mathbb{R}^m \longrightarrow \mathbb{R}$, or reports that the feasible region is empty.

**Algorithm 1** Bi-objective Epsilon-Constraint Method

```
Input: Objective bounds $\underline{f}, \overline{f} \in \mathbb{R}$ and increment $\delta \in \mathbb{R}$
1: $P := \emptyset$
2: $\epsilon := \overline{f}$
3: while $\epsilon \geq \underline{f}$ do
4:   $\mathbf{x} := opt(\mathbf{f}, \epsilon - \delta, \epsilon)$
5:   if $\nexists \mathbf{x}' \in P$ such that $\mathbf{x}' \succ \mathbf{x}$ then
6:     $P := P \cup \{\mathbf{x}\}$
7:   end if
8:   $\epsilon := \epsilon - \delta$
9: end while
Output: Set of Pareto-optimal decision vectors $P$
```

## 3 Drawbacks of the Original Epsilon-Constraint Method

Algorithm 1 gives an implementation according to the original description of the epsilon-constraint method from [6, p. 285] for the case of two objectives. The idea of the traditional epsilon-constraint method is to iteratively increase the constraint bound by a pre-defined constant $\delta$. The necessity to choose such a value represents also the main drawback of this approach. Since only one solution can be found in each interval, the discretization has to be fine enough not to "miss" any Pareto-optimal solution. In the worst case, the difference between objective vectors might be as small as the machine accuracy of the computer used to run the algorithm. Choosing such a small $\delta$, though, might cause a large number of redundant runs of the single-objective optimizer because they are constrained to an objective space subset which contains no Pareto-optimal objective vector. Thereby, a lot of search effort might be wasted and in fact prevents this method from being applicable to certain problems, as we demonstrate below. All these problems occur likewise in the epsilon-constraint metaheuristic proposed by [7], as the strategy to vary the parameters is essentially the same as in the original method.

The following example problem shows that the time complexity of the original epsilon-constraint method can be exponential in the problem size $n$, while the size of the Pareto set is only linear in $n$.

*Example 1.* The pseudo-Boolean function BBV : $\{0,1\}^n \to \mathbb{N}^2$ is defined as

$$\mathrm{BBV}(x_1, \ldots, x_n) = \left( \sum_{i=1}^{n} 2^{n-i} x_i, \quad \sum_{i=1}^{n} 2^{i-1}(1 - x_i) \right)$$

The example problem is a bi-objective generalization of the BinaryValue (BV) problem proposed in [8] for the complexity analysis of evolutionary algorithms. The name refers to the fact that a decision vector is the binary representation of the integer number given by its function value. Here, we use this BV function as the first objective, while the second objective is the BV function applied to the reversed and inverted sequence of bits.

By construction it is apparent that the BBV-function is actually a bijection from $X$ into $\mathrm{BBV}(X)$, the objective space $F$ contains $2^n$ distinct elements, hence $|X| = |F|$. However, its Pareto set contains exactly $n+1$ elements, which have the following simple description [4]: A decision vector $\mathbf{x} \in X$ is Pareto-optimal in the BBV problem, if and only if it has the form $1^k 0^{n-k}$, $k \in \{0, 1, \ldots, n\}$

**Proposition 1 ([4]).** *The expected running time of Algorithm 1 on the BBV problem is bounded below by* $\Omega(2^n \cdot T)$.

*Sketch of proof:* The proof is based on the observation that a $\delta \geq 4$ leads to missing at least one Pareto-optimal objective vector, regardless of the chosen bounds $\underline{f}, \overline{f}$. Therefore, $\delta < 4$ must be chosen, and at least $(2^n - 2)/4$ iterations are necessary to find all Pareto-optimal objective vectors. □

It could be argued that the problem here is only caused by an inappropriate definition of the constraint increments $\delta$. For our example problem, it would indeed be possible to define a different set of, e.g., $O(n)$ exponentially increasing constraint bounds that would lead to finding every single Pareto-optimal solution. But it is important to note that in a general scenario, where the solutions can be distributed arbitrarily in the objective space, such information is not available.

## 4 A New, Adaptive Epsilon-Constraint Method

Our idea to circumvent the deficit of the original epsilon-constraint method is to make use of information about the objective space as soon as it is available, and that is during the search process. This is the concept behind the new adaptive epsilon-constraint method described in this section.

In the two-objective case, the adaptive variation of the constraints is straightforward. There is only one constraint value to adapt, which can be achieved by starting with an arbitrary lower bound for $f_2$ and iteratively increasing the constraint on $f_2$ using the $f_2$-value of the optimum of the previous single-objective run. Such a scheme has been employed in [9] for generating all Pareto-optimal solutions for minimizing total cost and bottleneck time in a transportation problem. The three objective extension proposed in [9], however, is only able to find those Pareto-optimal solutions whose projection onto the $f_1$-$f_2$ plane equals the Pareto front of the two-objective problem. This indicates that the generalization of the scheme for higher objective space dimensions is more difficult and has therefore remained unsolved. The scheme we proposed in [4] and further elaborate here works for arbitrary numbers of objectives $m$. It is guaranteed to find all Pareto-optimal solutions for any $m$, provided that the underlying single-objective algorithm can solve the single-objective subproblems. For the special case of $m = 2$, our algorithm is equivalent to the scheme from [9].

The flowchart of the algorithm is depicted in Fig. 1. The core of the algorithm is an $m-1$ dimensional hypergrid, which partitions the whole objective

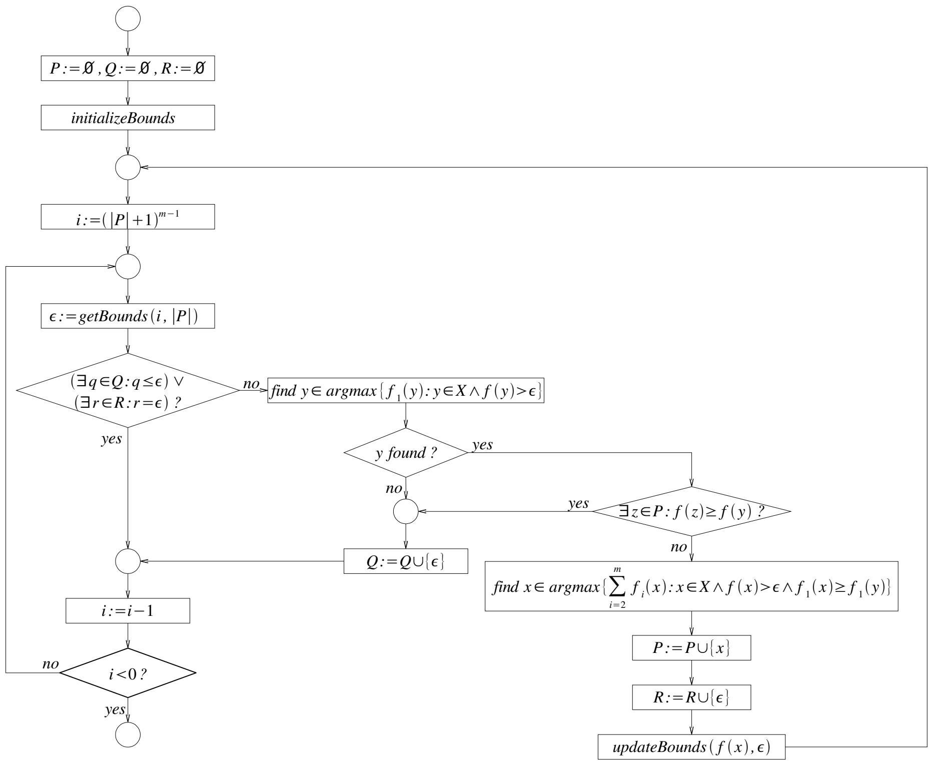

$P:=\emptyset, Q:=\emptyset, R:=\emptyset$
initializeBounds
$i:=(|P|+1)^{m-1}$
$\epsilon:=getBounds(i,|P|)$
$(\exists q \in Q: q \le \epsilon) \vee (\exists r \in R: r = \epsilon)$ ?
no
yes
find $y \in argmax\{f_1(y): y \in X \wedge f(y) > \epsilon\}$
y found ?
yes
no
$\exists z \in P: f(z) \ge f(y)$ ?
yes
no
$Q:=Q \cup \{\epsilon\}$
find $x \in argmax\{\sum_{i=2}^{m} f_i(x): x \in X \wedge f(x) > \epsilon \wedge f_1(x) \ge f_1(y)\}$
$i:=i-1$
$P:=P \cup \{x\}$
no
$i<0$ ?
yes
$R:=R \cup \{\epsilon\}$
updateBounds$(f(x),\epsilon)$

**Fig. 1.** Flowchart of the new adaptive epsilon-constraint algorithm

space into rectangular axis-parallel co-domains. The coordinates for this grid are determined by the function values of the already identified Pareto-optimal solutions. These coordinates are stored in the matrix $\mathbf{e} = (\mathbf{e_1}, \ldots, \mathbf{e_m})$, where the $\mathbf{e_i}$ are vectors containing the grid coordinates for objective $i$, defined by the $f_i$-values of all Pareto-optimal solutions found so far. These vectors $\mathbf{e_i}$ initially contain only suitable lower bounds, which can be guaranteed to be smaller than all values of the respective objective component. In this case we chose a lower bound of zero assuming a non-negative objective space.

In each iteration of the outer loop, one new Pareto-optimal point is sought. This is achieved by performing two consecutive constrained single-objective optimization runs for each grid point as a lower bound in decreasing order of the index $i$. After a new Pareto-optimal point $\mathbf{x}$ is found, this point is added to the set of already found solutions, $P$, and the grid is updated by recording the objective values of $\mathbf{x}$. For each objective $j \in \{1, \ldots, m\}$, the value $f_j(\mathbf{x})$ is inserted into the sorted vector $\mathbf{e_j}$. If no feasible solution can be found or the solution is dominated by any previously found solution, the searched objective co-domain is marked by storing the lower bound vector $\epsilon$ in the set $Q$. The purpose of function *getBounds* is to translate the iteration counter $i$ into the $m-1$ corresponding

```
Function initializeBounds
1: for $j := 1$ to $m$ do
2:     $\mathbf{e}_j := (0)$
3: end for
```

```
Function getBounds$(i, p)$
1: $\epsilon_1 := 0$
2: for $j := 2$ to $m$ do
3:     $d := i \bmod (p + 1)$
4:     $i := (i - d) \,/\, (p + 1)$
5:     $\epsilon_j := e_{j_d}$                                  {lower bound for objective $j$}
6: end for
7: return $\epsilon$
```

```
Function updateBounds$(\mathbf{y})$
1: for $j := 1$ to $m$ do
2:     $i := 1$
3:     while $e_{j_i} < y_j$ do
4:         $i := i + 1$                                  {search for insertion position}
5:     end while
6:     $\mathbf{e}_j := (e_{j_1}, \ldots e_{j_{i-1}}, y_j, e_{j_{i+1}}, \ldots, e_{j_{|P|+2}})$     {insert new constraint value}
7: end for
```

**Fig. 2.** Subroutines for storing, sorting and retrieving the bounds

indices to retrieve the right constraint values from each of the $\mathbf{e_i}$ vectors. The total number of calls to the single-objective optimization algorithm can now be bounded by $(k+1)^{m-1}+k$, where $k$ is the cardinality of the Pareto front. Using as upper bound $T$ on the running time of the single-objective algorithm yield the following result.

**Theorem 1.** *The running time of the algorithm given in Fig. 1 to discover a Pareto front of an $m$-objective problem with $k$ elements is at most $T[(k+1)^{m-1}+k]$, where $T$ is the running time of the single-objective optimization algorithm.*

*Sketch of proof:* To show that the image of $P$ at the end of the run equals the Pareto front $F^*$, it has to be shown that (i) only Pareto-optimal solutions enter $P$ and (ii) no Pareto-optimal objective vector is missed. For (i) we observe that $x$ can only be dominated by solutions from regions which has already been fully searched and the dominance check would then prevent $x$ from entering $P$. For (ii) it can be shown that any potentially missed Pareto-optimal objective vector must necessarily be the image of the optimum $y$ of some grid cell, and was therefore already found. The bound on the number of calls to the single-objective optimization algorithm is due to the observation that in each iteration at least one region is marked as searched, and the maximum number of different regions to be searched is bounded by the maximum number of grid points, which is $(k+1)^{m-1}$. In addition, the second run of the single-objective optimization is only performed if a new solution is found, which gives leads to an additional number of $k$ runs. □

## 5 Simulation Results

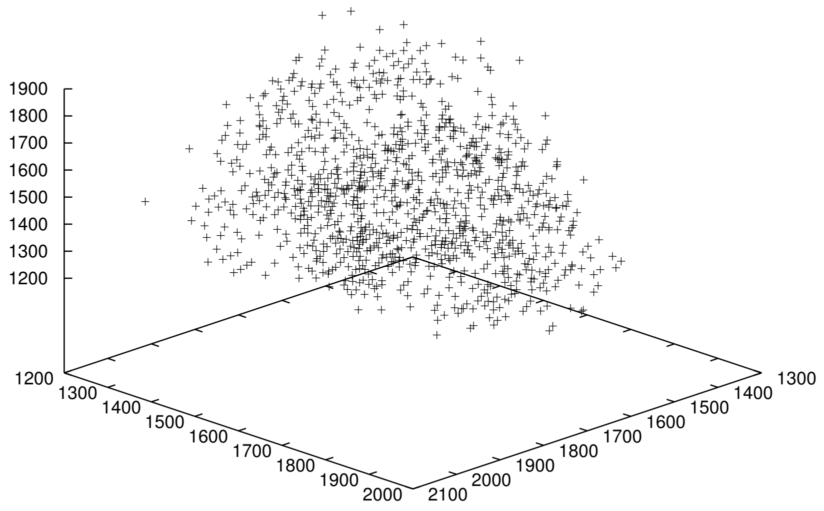
This section presents some simulation results of the new algorithm on the multiobjective knapsack problem. The multiobjective knapsack problem is one of the most extensively used benchmark problem for multiobjective metaheuristics (see, e.g., [10], [11], or [7]). Given is a set of $n$ items, each of which has $m$ profit and $k$ weight values associated with it. The goal is to select a subset of items such that the sums over each of their $k$-th weight values do not exceed given bounds and the sums over each of their $m$-th profit values are maximized. A representation as a pseudo-Boolean optimization problem is typically used, where the $n$ binary decision variables denote whether an item is selected or not. For empirical studies, the parameters of the problem, the weight and profit values, are usually drawn at random from a given probability distribution. Here, the weights and profits are randomly chosen integers between 10 and 100, and the capacities are set to half of the sum of the weights.

Table 1 summarizes the results obtained for different instances of the knapsack problem with 3 objectives. [3] To the best of our knowledge no exact Pareto fronts have been computed so far for the three-objective knapsack problem, probably due to the lack of an appropriate generating method besides complete enumeration. Note that the total number of single-objective runs given in Table 1 is considerably lower than the upper bound given in Theorem 1. Fig. 3 gives a visual impression of the obtained three-dimensional Pareto fronts.

**Table 1.** Results on different instances of the knapsack problem with three objectives

| n | 10 | 20 | 30 | 40 | 50 | 100 |
|---|---|---|---|---|---|---|
| single-objective runs | 76 | 1622 | 9870 | 26846 | 128695 | 644689 |
| total CPU time | 4 sec | 102 sec | 20 min | 62 min | 445 min | 255 h |
| $\|P\|$ | 9 | 61 | 195 | 389 | 1048 | 6501 |

[3] The source code of the algorithm implemented in C and invoking CPLEX as an example for an arbitrary single-objective optimization technique is available from `http://www.tik.ee.ethz.ch/~laumanns` as well as the data of the knapsack problem instances and the Pareto fronts from `http://www.tik.ee.ethz.ch/~zitzler/testdata.html`.



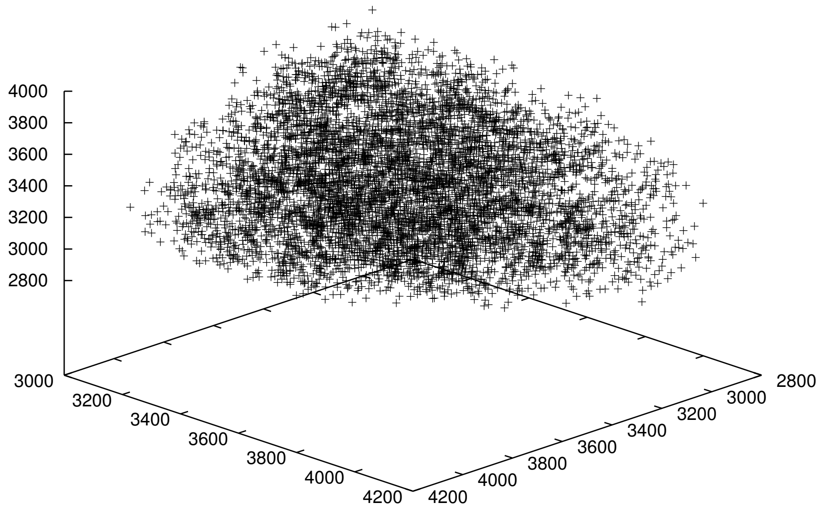


**Fig. 3.** Knapsack problem: plots of the exact Pareto fronts for $m = 3$ objectives, $n = 50$ (top) and $n = 100$ (bottom).

## 6 Conclusions and Theoretical Implications

We have presented a new scheme for generating or approximating the Pareto set of multiobjective optimization problems based on the well-known epsilon-constraint method. Its time complexity, measured by the number of constrained single-objective sub-problems to be solved, is $O(k^{m-1})$, where $k$ is the cardinality of the Pareto front to be found and $m$ the number of objectives. The implementation we have provided is compact, which facilitates its use in practice.

The existence of such a scheme with a polynomially bounded number of single-objective sub-problems suggests the following implications regarding the computational complexity of single- and multiobjective optimization. It is well-known that if the constrained single-objective problem (cSOP) is NP-hard then the corresponding multiobjective problem (MOP) is NP-hard as well. Not much attention has been paid to the reverse, however. Under what circumstances can we state that if cSOP is 'easy' (i.e., solvable in polynomial time), the corresponding MOP is also easy (solvable in polynomial time)? We have of course to exclude cases, where the Pareto front is exponentially large. But if we assume that Pareto front contains a polynomially bounded number of elements, or are only looking for a polynomially bounded subset [12] as an approximate solution set, things change. The algorithms presented in this paper and in [4] yield the proof that in this case solving MOP is not fundamentally more difficult than cSOP and, as a direct consequence, if the MOP was still NP-hard then the cSOP without restrictions on the range of constraint values would be NP-hard as well.

The proposed algorithm is also useful when the aim is to find only a representative subset of all Pareto-optimal solutions, for example in continuous objective spaces. The only necessary change to adapt to such a scenario is to introduce appropriate increments for the constraint values, like in the original epsilon-constraint method. But in contrast to its predecessor, the new algorithm will still avoid sampling in empty objective space regions to a large extent and hence be more efficient in many cases, e.g., for continuous problems where the Pareto front contains several disconnected parts.

The algorithm has been designed as a generic framework for different single-objective optimizers. In case of problems where no efficient single-objective optimizer is available, one could use a heuristic or approximative method instead. This could be especially useful, when a good single-objective heuristic is available, but it is not clear how to extend the heuristic to handle multiple objectives simultaneously. In other cases, where true multiobjective metaheuristics are available, the resulting algorithm would constitute a baseline algorithm, which can be compared with other multiobjective metaheuristics. The necessity for such baseline algorithms has been pointed out by many researchers in the field.

A further application area of the results of this paper is the running time analysis of both exact and heuristic methods for specific problems or problem instances. It is only necessary to instantiate our algorithm with an appropriate method to solve the constrained sub-problems and to derive an upper bound of the running time for this specific method on the given problem. Thereby, a problem-specific algorithm is obtained together with an upper running time bound, which also can serve as a baseline to judge the efficiency of other methods.

Finally, a challenging and so far unsolved question is whether it is possible to get rid of the exponent in the running time bound and to prove an upper bound which is linear in the number of desired Pareto-optimal objective vectors.

## References

1. Hwang, C.L., Masud, A.S.M.: Multiple Objectives Decision Making—Methods and Applications. Springer, Berlin (1979)
2. Miettinen, K.: Nonlinear Multiobjective Optimization. Kluwer, Boston (1999)
3. Ehrgott, M.: Multicriteria optimization. Springer, Berlin (2000)
4. Laumanns, M., Thiele, L., Zitzler, E.: An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. European Journal of Operational Research (2004) In press. Accepted 13 August 2004. Available online 17 November 2004.
5. Haimes, Y., Lasdon, L., Wismer, D.: On a bicriterion formulation of the problems of integrated system identification and system optimization. IEEE Transactions on Systems, Man, and Cybernetics **1** (1971) 296 – 297
6. Chankong, V., Haimes, Y.: Multiobjective Decision Making Theory and Methodology. Elsevier (1983)
7. Ranjithan, S.R., Chetan, S.K., Dakshina, H.K.: Constraint Method-Based Evolutionary Algorithm (CMEA) for Multiobjective Optimization. In Zitzler, E., et al., eds.: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001). Volume 1993 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2001) 299–313
8. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science **276** (2002) 51–81
9. Srinivasan, V., Thompson, G.L.: Algorithms for minimizing total cost, bottleneck time and bottleneck shipment in transportation problems. Naval Research Logistics Quarterly **23** (1976) 567–598
10. Zitzler, E., Thiele, L.: Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. IEEE Transactions on Evolutionary Computation **3** (1999) 257–271
11. Jaszkiewicz, A.: On the performance of multiple objective genetic local search on the 0/1 knapsack problem. a comparative experiment. IEEE Transactions on Evolutionary Computation **6** (2002) 402–412
12. Laumanns, M., Thiele, L., Deb, K., Zitzler, E.: Combining convergence and diversity in evolutionary multiobjective optimization. Evolutionary Computation **10** (2002) 263–282