Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 169 (2006) 932–942
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method

Marco Laumanns [a,*], Lothar Thiele [b], Eckart Zitzler [b]

[a] *Swiss Federal Institute of Technology Zurich (ETH), Institute for Operations Research, Clausiusstrasse 47, ETH-Zentrum, CH-8092 Zurich, Switzerland*
[b] *Swiss Federal Institute of Technology Zurich (ETH), Computer Engineering and Networks Laboratory, Gloriastrasse 35, ETH-Zentrum, CH-8092 Zurich, Switzerland*

Accepted 13 August 2004

Abstract

This paper discusses methods for generating or approximating the Pareto set of multiobjective optimization problems by solving a sequence of constrained single-objective problems. The necessity of determining the constraint value a priori is shown to be a serious drawback of the original epsilon-constraint method. We therefore propose a new, adaptive scheme to generate appropriate constraint values 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. We prove that—independent of the problem or the problem size—the time complexity of the new scheme is $O(k^{m-1})$, where k is the number of Pareto-optimal solutions to be found and m the number of objectives. Simulation results for the example problem as well as for different instances of the multiobjective knapsack problem demonstrate the behavior of the method, and links to reference implementations are provided.

Keywords: Multiple objective optimization; Non-dominated set; Pareto set; Epsilon-constraint method; Generating methods

1. Introduction

An important task in multiobjective optimization is the identification of 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 sometimes desirable

* Corresponding author.
E-mail addresses: laumanns@ifor.math.ethz.ch (M. Laumanns), thiele@tik.ee.ethz.ch (L. Thiele), zitzler@tik.ee.ethz.ch (E. Zitzler).

0377-2217/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2004.08.029

to identify *all* Pareto-optimal solutions. This maximizes the choice 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. Thus, methods to generate the Pareto set, e.g., for simple benchmark problems or small instances of difficult problems, are useful auxiliary tools.

Whenever a multiobjective problem has a finite number of Pareto-optimal solutions, 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 [7,10,5]. 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. This must be taken with caution, however. The weighted sum method, for example, only guarantees to find supported Pareto-optimal solutions, i.e., those lying in convex regions of the objective space. Non-supported solutions can only be found coincidently when they are discovered as intermediate solutions on the way towards a supported solution. The epsilon-constraint method [6] works by pre-defining a virtual grid in the objective space and solving different single-objective problems constrained to each grid cell. All Pareto-optimal solutions can be found only if this grid is fine enough such that at most one Pareto-optimal solution is contained in each cell. For a general problem, the choice of the grid size parameter is therefore not only very difficult, it also influences the running time of the whole algorithm.

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 meta-strategies 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.

To the best of our knowledge, no scheme is available that can determine the whole Pareto set by a number of single-objective subproblems which depends only on the number of Pareto-optimal solutions and not on additional properties such as the distribution of the Pareto-optimal objective vectors. The purpose of this paper is to present such a scheme based on the epsilon-constraint method. We prove 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 [6] 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 we will show 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.

Our method is also applicable when no efficient algorithm to solve the constrained single-objective problems is available, e.g., when these problems are NP-hard. In such cases, a heuristic method can be used to obtain an approximate solution. Our method, used with a heuristic to solve the resulting single-objective problems, thereby results in a metaheuristic whose aim is to find a set of approximate Pareto-optimal solutions accessible to the single-objective heuristic within a given fixed running time bound.

2. Problem scenario

The problem we are dealing with in this paper is to find or to approximate the Pareto set of a general multiobjective problem with m objectives. We

assume that all objective functions are to be maximized.

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 $\mathbf{x}^* \in X$ *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 $f_i(\mathbf{x}) \geqslant f_i(\mathbf{x}^*)$ for all $i = 1, \ldots, m$ and $f_i(x) > f_i(x^*)$ 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 [5].

Several methods exist devoted to this task, usually referred to as "a posteriori" or "non-dominated solution generation" methods [7]. 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 [6]. 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, here referred to as $opt(\mathbf{f}, \epsilon, \epsilon')$. The details of this procedure are not relevant for the consideration of this study. We simply assume that it terminates after a fixed time T and returns either the optimum of the constrained problem $(\mathbf{f}, \epsilon, \epsilon')$

$$\text{lex max} \quad \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})) \tag{1}$$

$$\text{subject to} \quad \epsilon_i < f_i(\mathbf{x}) \leqslant \epsilon'_i \quad \forall i \in \{2, \ldots, m\} \tag{2}$$

$$\mathbf{x} \in X \tag{3}$$

(if it exists), an approximation of it, or null (if the feasible region is empty). The notation 'lex max' refers to lexicographic maximization of the m different objectives. The reason for this is that there might be several solutions with maximum f_1-value. Out of these, we want to find the one with maximum f_2-value, and so forth. In contrast, if the single-objective optimizer stops already after maximizing only the first objective, the resulting solution is only guaranteed to be weakly Pareto-optimal [5]. If the given single-objective optimizer is only able to solve a standard constrained single-objective problem, it can easily adjusted to solve the above 'lex max' problem by m runs successively optimizing the different objectives, while in each run the obtained optimal value is added as a constraint. If T' is an upper bound for the running time of the single-objective runs, an upper bound of $T = mT'$ for the $opt(\mathbf{f}, \epsilon, \epsilon')$ procedure can be assumed.

3. Drawbacks of the original epsilon-constraint method

Algorithm 1 gives an implementation according to the original description of the epsilon-constraint method from [3, 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 δ. 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

Algorithm 1. Bi-objective Epsilon-Constraint Method

Input: Objective bounds $\underline{f}, \bar{f} \in \mathbb{R}$ and increment $\delta \in \mathbb{R}$

```
P := ∅
ε := f̄
while ε ≥ f̲ do
    x := opt(f, ε − δ, ε)
    if ∄ X' ∈ P such that x' ≻ x then
        P := P ∪ {x}
    end if
    ε := ε − δ
end while
```

Output: Set of Pareto-optimal decision vectors P

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 δ, 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 [11], 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 $\mathrm{BBV} : \{0,1\}^n \to \mathbb{N}^2$ is defined as

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

The example problem is a bi-objective generalization of the BINARY VALUE (BV) problem proposed in [4] 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 contain 2^n distinct elements, hence $|X| = |F|$. However, its Pareto set contains exactly $n+1$ elements, which have the following simple description:

Proposition 1. *A decision vector $\mathbf{x} \in X$ is Pareto-optimal in the BBV problem, if and only if it has the form 1^k0^{n-k}, $k \in \{0,1,\ldots,n\}$.*

Proof. "⇒": Assume that a Pareto-optimal decision vector $\mathbf{x}$ does not have the claimed form. This means that there is a k such that $x_k = 0$ and $x_{k+1} = 1$. If we exchange these two bits, both objective values are increased, therefore x cannot be Pareto-optimal.

"⇐": Assume $\mathbf{x}$ has the form 1^k0^{n-k} and is dominated by $\mathbf{x}' \in X$. Either the first or the second objective value of $\mathbf{x}'$ must be larger than the respective objective value of $\mathbf{x}$. The relation $f_1(\mathbf{x}') > f_1(\mathbf{x})$ can only be achieved by keeping the k ones fixed and replacing an arbitrary number of the zeroes by ones, which implies $f_2(\mathbf{x}') < f_2(\mathbf{x})$. The relation $f_2(\mathbf{x}') > f_2(\mathbf{x})$ can only be achieved by keeping the $n-k$ zeroes fixed and replacing an arbitrary number of the ones by zeros, which implies $f_1(\mathbf{x}') > f_1(\mathbf{x})$. In both cases, $\mathbf{x}'$ does not dominate $\mathbf{x}$, which contradicts the assumption. □

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

Proof. Consider the three Pareto-optimal decision vectors $\mathbf{a} = 0^n$, $\mathbf{b} = 10^{n-1}$ and $\mathbf{c} = 110^{n-2}$. The corresponding objective vectors are $\mathbf{f}(\mathbf{a}) = (0, 2^n - 1)$, $\mathbf{f}(\mathbf{b}) = (2^{n-1}, 2^n - 2)$ and $\mathbf{f}(\mathbf{c}) = (2^{n-1} + 2^{n-2}, 2^n - 4)$. Assume a $\delta \geqslant 4$ is chosen and denote with $\epsilon^{(t)}$ the value of the current (upper) constraint in iteration t of the algorithm. We will show that

there no way to select a starting value $\bar{f}$ such that the three objective vectors $\mathbf{f}(\mathbf{a})$, $\mathbf{f}(\mathbf{b})$ and $\mathbf{f}(\mathbf{c})$ are found in different iterations of the while-loop of the algorithm: either the decision vectors **a** and **b** or **b** and **c** will fall into the same interval. In order for a being feasible, the current constraint bound must be $\epsilon^{(t)} \in [2^n - 1, 2^n - 1 + \delta]$. Now consider the two cases $\epsilon^{(t)} \in [2^n - 1, 2^n - 2 + \delta]$ and $\epsilon^{(t)} \in [2^n - 1, 2^n - 1 + \delta]$. If $\epsilon^{(t)} \in [2^n - 1, 2^n - 1 + \delta]$, **b** will also be a feasible solution in this iteration and, as it has a larger f_1-value than **a**, **a** will not be returned by opt in this iteration. In the next iteration, $\epsilon^{(t+1)} < 2^n - 2 < f_2(\mathbf{a})$, hence **a** is infeasible and will never be found. If $\epsilon^{(t)} \in [2^n - 2, 2^n - 1 + \delta]$, then in the next iteration $\epsilon^{(t+1)} \in [2^n - 2, 2^n - 1]$. In this case, **c** is also feasible, and, as it has a larger f_1 value than **b**, **b** will not be returned by opt in this iteration. In the subsequent iteration, $\epsilon^{(t+2)} < 2^n - 5 < f_2(\mathbf{b})$, hence **b** is infeasible and will never be found. 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 δ. 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 [12] for generating all Pareto-optimal solutions for minimizing total cost and bottleneck time in a transportation problem. The three objective extension proposed in [12], 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 propose in this paper 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 [12].

The new algorithm is given in pseudo-code as Algorithm 2. The core of the algorithm is an $m - 1$ dimensional hypergrid, which partitions the whole objective space into rectangular axis-parallel co-domains with respect to objectives f_2 to f_m. 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 $\mathbf{e} = (\mathbf{e_2}, \ldots, \mathbf{e_m})$, 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 minus and plus infinity, so that the initial grid is composed of only one cell, the whole potential objective space, $\mathbb{R}^m$.

In each iteration of the outer loop (line 2–13), one new Pareto-optimal point is sought. This is achieved by performing a constrained single-objective optimization run for each grid cell (line 7) in decreasing order of the cell index i. After a new Pareto-optimal point **x** is found, this point is added to the set of already found solutions, P, and the grid is updated by including the objective values of x. For each objective $j \in \{2, \ldots, m\}$, the value $f_j(\mathbf{x})$ is inserted into the sorted vector $\mathbf{e_i}$. If no feasible solution can be found or the solution is dominated by any previously found solution, the

Algorithm 2. Adaptive m-objective Epsilon-Constraint Method

1: $P := \emptyset$; $Q := \emptyset$; $\mathbf{e}_j := (-\infty, \infty)\ \forall 2 \leqslant j \leqslant m$
2: $i := (|P|+1)^{m-1}$ {initialize subregion counter}
3: $i := i - 1$
4: **if** $i < 0$ **then stop** {no new solution found}
5: $(\epsilon, \epsilon') := getConstraints(i, \mathbf{e}, P)$
6: **if** $[\epsilon, \epsilon'] \subset Q$ **then goto** 3 {if subregion already searched}
7: $\mathbf{x} := \text{opt}(f, \epsilon, \epsilon')$ {solve single-objective problem}
8: **if** $\mathbf{x} = null$ **then** $Q := Q \cup [\epsilon, \epsilon']$; **goto** 3 {if subregion empty}
9: **if** $\exists\, y \in P : y \succ x$ **then** $Q := Q \cup [\epsilon, \epsilon']$; **goto** 3 {if solution dominated}
10: $P := P \cup \{x\}$
11: $Q := Q \cup [\epsilon, \mathbf{f}(x)]$
12: $\mathbf{e} :=$ *updateConstraints* $(\mathbf{f}(\mathbf{x}), \mathbf{e})$
13: **goto** 2 {new Pareto-optimal solution was found}

Output: Set of Pareto-optimal decision vectors P

Function *getConstraints* $(i, \mathbf{e}, P)$
1: **for** $j := 2$ to m **do**
2: $d := i \bmod (|P|+1)$
3: $i := (i-d)/(|P|+1)$
4: $\epsilon_j := e_{j_d}$ {lower constraint on objective j}
5: $\epsilon'_j := e_{j_{d+1}}$ {upper constraint on objective j}
6: **end for**
7: **return** (ϵ, ϵ')

Function *updateConstraints* $(\mathbf{y}, \mathbf{e})$
1: **for** $j := 2$ 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: $e_j := (e_{j_1}, \ldots, e_{j_{i-1}} y_j, e_{j_{i+1}}, \ldots, e_{j_{|P|+2}})$ {insert new constraint value}
7: **end for**
8: **return** $\mathbf{e} := (\mathbf{e}_2, \ldots, \mathbf{e}_m)$

region $[\epsilon, \epsilon'] := \{\mathbf{y} \in \mathbb{R}^m \mid \forall 2 \leqslant j \leqslant m : \epsilon_j \leqslant y_j \leqslant \epsilon'_j\}$, i.e., the cuboid between the vectors ϵ and ϵ' is added to the set of already searched regions Q. The purpose of function *getConstraints* is to translate the iteration counter i into the $m-1$ corresponding indices to retrieve the right constraint values from each of the $\mathbf{e_i}$ vectors. The working principle of the algorithm is depicted in Fig. 1.

Theorem 1. *The running time of Algorithm 2 to discover a Pareto front of an m-objective problem with k elements is* $O(k^{m-1} \cdot T)$, *where T is the running time of the single-objective optimizer.*

Proof. Let K denote the number of times that the single-objective optimizer will be called during the

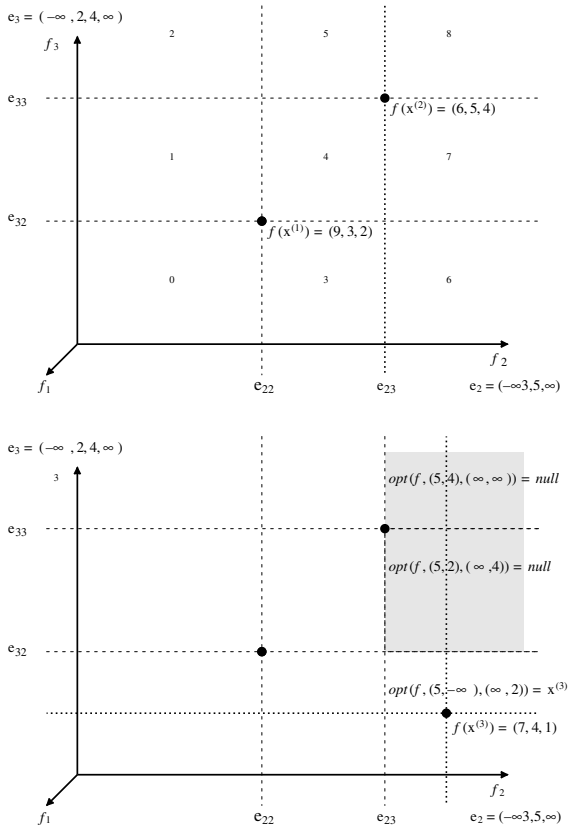

Fig. 1. Working principle of the adaptive epsilon-constraint algorithm. The top diagram shows the state of the algorithm at the beginning of the third iteration of the outer loop (line 2). So far, two Pareto-optimal solutions, $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, have been found, and the constraint vectors have $|P| + 2 = 4$ components each, dividing the objective space into nine subspaces. The grid cells correspond to the projection of these subspaces into the f_2–f_3 plane. They are numbered according to the corresponding iteration counter i. Starting will cell 8, the single-objective optimizer uses the respective upper and lower constraint values. The hypothetical results are depicted in the lower diagram. For cells 8 and 7, the optimizer returns *null* indicating that this region is empty. For cell 6, the optimizer returns a new Pareto-optimal point $\mathbf{x}^{(3)}$ and the loop is finished (jump back to line 2). The two additional constraint values defined by $\mathbf{f}(\mathbf{x}^{(3)})$ are indicated with dotted lines.

run of the algorithm. Our aim is to bound K. The procedure *opt* is invoked with different combinations of upper and lower constraint vectors (ϵ, ϵ'). For a given constraint matrix $\mathbf{e}$, the number of different combinations equals $(|P| + 1)^{m-1}$, as used in line 2 to initialize the iteration counter. This number is maximal at the end of the run, when $\mathbf{f}(P) = F^*$, i.e., the whole Pareto front has been discovered. For each current value of $|P|$, the constraint combinations represent a partition of the search space. In addition, each new objective vector leads to the inclusion of $m - 1$ new constraint values and therefore a subdivision of some of the previous constraint regions. Since the searched regions are monitored in the set Q, line 3 guarantees that *opt* will not be invoked for any region that is contained in Q and thus marked as empty. Hence, *opt* cannot be invoked for more than $(|F^*| + 1)^{m-1}$ times. With T being an upper bound on the running time of *opt*, the claimed bound for the total running time follows.

It remains to be shown (1) that only Pareto-optimal solutions will enter the set P and (2) that no Pareto-optimal solution will be missed, i.e., there is no Pareto-optimal objective vector that is not represented by some element of P when the algorithm terminates. For (1) assume that $\mathbf{x}$ is not Pareto-optimal. As $\mathbf{f}(\mathbf{x})$ is the lexicographically maximal objective vector from the current grid cell, $\mathbf{x}$ is not dominated by any other element from this cell. All elements dominating $\mathbf{x}$ must therefore either belong to the same grid cell, which was already ruled out, or to a grid cell with all lower constraint bounds at least as large as those of the current grid cell. However, the decrementing of the counter i together with the mechanism to calculate the grid coordinate indices in function *getConstraints* guarantees that all such grid cells have already been investigated previously to the current grid cell. Since adding a new point to P causes the reset of the iteration counter (jump from line 13 to line 2), all Pareto-optimal solutions from these cells are already contained in P. If $\mathbf{x}$ is dominated by any of these solutions, line 9 prevents it from being included in P. For (2) assume that there is some $g \in F^*$ for which there is no $\mathbf{x} \in P$ with $\mathbf{f}(\mathbf{x}) = g$ and the algorithm terminates. This means that $i < 0$ in line 4, which implies that for all i with $(|P| + 1)^{m,-1} > i \geqslant 0$, *opt* either returned *null* or a solution dominated some element of P, or was not executed because the corresponding region is already contained in Q. Since the union of the regions for all i with $(|P| + 1)^{m-1} > i \geqslant 0$ represents a partition of the

whole search space, some of it must contain an y with $\mathbf{f}(y) = g$, which will be found by a call to *opt* for this region. Hence, a contradiction is reached, which completes the proof. □

5. Simulation results

This section presents some simulation results of the new algorithm on the example function BBV as well as on some instances of the multiobjective knapsack problem.

5.1. The BBV Problem

For the BBV problem, we use Algorithm 2 as a metaheuristic. An evolutionary algorithm (EA) is used for the single-objective subproblems, in this case a simple $(\mu + \lambda)$-EA. In the EA, the n decision variables are represented as a binary vector with a mutation rate of $1/n$ and no recombination. A population size of $\mu = \lambda = n + 1$ was used. Each single-objective run was terminated after $10n$ generations. This value was determined experimentally such that for each instance, the whole Pareto set was generated in the majority of the trials. Table 1 compares the number of single-objective runs used by the new algorithm to those used by the original method. The parameters for the original method (Algorithm 1) were $\underline{f} = -1$, $\overline{f} = 2^n - 1$ and $\delta = 1$. The results match with the predictions of Proposition 2 and Theorem 1 and demonstrate the immense (linear versus exponential) difference with respect to the dependency of the running time on the problem size n. The experiments for the original method were therefore only performed up to $n = 20$. The achieved Pareto fronts are plotted in Fig. 2 for different values of n. For better visualization, the axis have been transformed according to $\mathrm{BBV}_i \mapsto \log(2^n - \mathrm{BBV}_i)$.

Table 1
BBV problem: Number of single-objective runs used by the adaptive (Algorithm 2) and by the original epsilon-constraint method (Algorithm 1)

n	$\lvert F^* \rvert$	Algorithm 2	Algorithm 1
2	3	4	5
4	5	6	17
10	11	12	1025
20	21	22	1048577
40	41	42	
80	81	82	

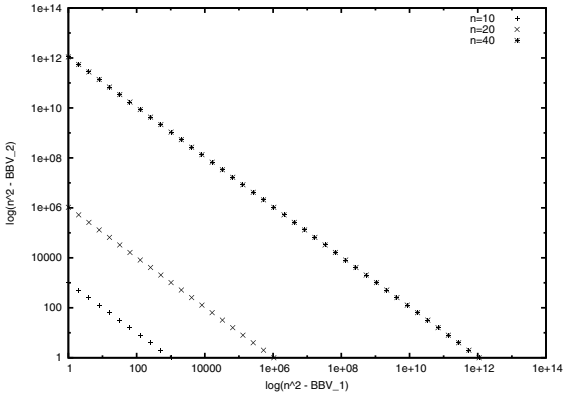

Fig. 2. BBV problem: Pareto fronts for $n = 10$, 20, and 40. The objective values have been transformed according to $\mathrm{BBV}_i \mapsto \log(2^n - \mathrm{BBV}_i)$.

5.2. The multiobjective knapsack problem

The multiobjective knapsack problem is one of the most extensively used benchmark problem for multiobjective metaheuristics (see, e.g., [13,9,11]). 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 kth weight values do not exceed given bounds and the sums over each of their mth 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.

For this problem we used the Algorithm 2 both as a metaheuristic to approximate the Pareto set and as an exact method to generate the whole Pareto set. The metaheuristic version again uses

Table 2
Knapsack problem: Comparison of the results for the exact method using an ILP-solver (upper section) with the metaheuristic using a single-objective EA (lower section)

	n	10	20	30	40	50
ILP-solver	Single-objective runs	76	1618	9867	26,839	128,676
	total CPU time	2 minutes	37 minutes	2 hours	4 hours	24 hours
	$\|P\|$ (note: $P = F^*$)	9	61	195	389	1024
EA	Single-objective runs	84	398	1520	2505	14,496
	generations per run	20	40	60	80	100
	$\|P\|$	10	27	66	84	209
	$\|P \cap F^*\|$	7	23	16	25	45
	$\frac{\|P \cap F^*\|}{\|F^*\|}$	77.8%	37.7%	8.2%	6.4%	4.4%

the EA described above as the underlying single-objective optimization algorithm. Differently to the BBV problem, uniform crossover is now used in the EA and the mutation rate was set to $2/n$. The population size was set to $\mu = \lambda = n$. The exact version employs CPLEX [1,8], thereby solving the resulting integer linear program (ILP) exactly via a branch-and-bound algorithm.

Table 2 summarizes the results obtained for different instances of the knapsack problem with three objectives. While the exact version only requires one run of the whole algorithm for each instance, the EA results are averaged over 10 independent trials. It can be seen that the size of the Pareto set, and therefore the total running time of the exact method, increases very quickly with the problem size, which makes it very difficult to solve larger instances of the three-objective knapsack problem to optimality. The empirical test also shows that the running time of the branch-and-bound algorithm increases considerably in later iterations, when the objective space becomes more dense and the constraint bounds become very tight. Nevertheless, it is noteworthy that 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 2 is also lower than the upper bound given in Theorem 1. By choosing reasonable running times, the metaheuristic can of course be applied to arbitrarily large problems. The results indicate, however, that it has increasing difficulty to find Pareto-optimal solutions: the *coverage* value [13], i.e., the percentage of discovered Pareto-optimal solutions, decreases rapidly. Fig. 3 gives a visual impression of the obtained three-dimensional Pareto fronts.

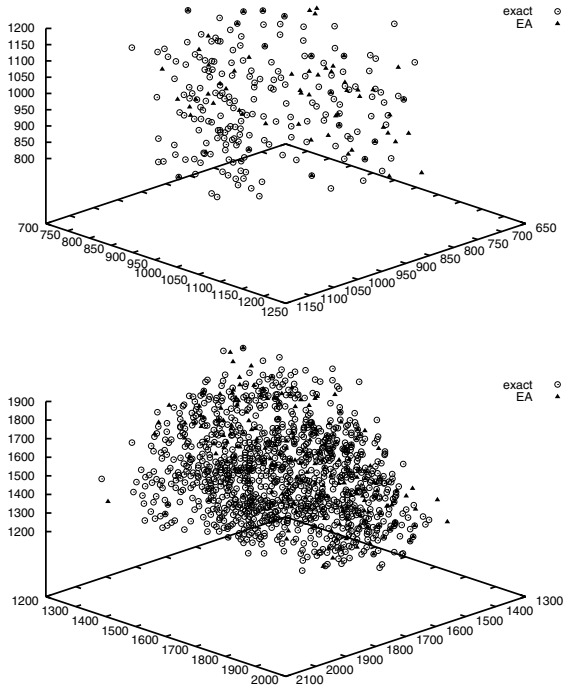

Fig. 3. Knapsack problem: plots of the obtained Pareto fronts for $m = 3$ objectives, $n = 30$ (top) and $n = 20$, $n = 50$ (bottom), both methods.

5.3. Implementation notes

Both the metaheuristic and the exact version of Algorithm 2 have been implemented in C and are available for download to facilitate their use for other problems and to encourage further comparative studies.

The metaheuristic using the single-objective evolutionary algorithm has been implemented as an independent module within the PISA [2] framework and is therefore also available as ready-to-use executables for various operating systems. Besides the BBV and the knapsack problems, practically all test functions available from the PISA website [1] can be used with it. The source code for the exact version invoking CPLEX as an example for an arbitrary single-objective optimization technique is also available, [2] as well as the data of the knapsack problem instances. [3]

6. Conclusion

We have presented a new metaheuristic scheme for generating or approximating the Pareto set of multiobjective optimization problems based on the well-known epsilon-constraint method. We have shown that 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 number of Pareto-optimal solutions to be found and m is the number of objectives. The implementation we have provided is compact, which facilitates its use in practice.

The proposed 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 theoretical and thus 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.

[1] http://www.tik.ee.ethz.ch/pisa.

[2] http://www.tik.ee.ethz.ch/~laumanns.

[3] http://www.tik.ee.ethz.ch/~zitzler/testdata.html.

References

[1] R.E. Bixby, M. Fenelon, Z. Gu, E. Rothberg, R. Wunderling, MIP: Theory and practice—closing the gap, in: M.J.D. Powell, S. Scholtes (Eds.), System Modelling and Optimization: Methods, Theory and Applications, Kluwer, 2000, pp. 19–50.

[2] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler, PISA—a platform and programming language independent interface for search algorithms. In: Evolutionary Multi-Criterion Optimization (EMO 2003), Lecture Notes in Computer Science, Springer, Berlin, 2003.

[3] V. Chankong, Y. Haimes, Multiobjective Decision Making Theory and Methodology, Elsevier, 1983.

[4] S. Droste, T. Jansen, I. Wegener, On the analysis of the $(1+1)$ evolutionary algorithm, Theoretical Computer Science 276 (1–2) (2002) 51–81.

[5] M. Ehrgott, Multicriteria Optimization, Springer, Berlin, 2000.

[6] Y. Haimes, L. Lasdon, D. Wismer, 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.

[7] C.-L. Hwang, A.S.M. Masud, Multiple Objectives Decision Making Methods and Applications, Springer, Berlin, 1979.

[8] ILOG, Gentilly, France, ILOG CPLEX 7.0 User's Manual, 2000.
[9] A. Jaszkiewicz, On the performance of multiple objective genetic local search on the 0/1 knapsack problem, a comparative experiment, IEEE Transactions on Evolutionary Computation 6 (4) (2002) 402–412.
[10] K. Miettinen, Nonlinear Multiobjective Optimization, Kluwer, Boston, 1999.
[11] S.R. Ranjithan, S.K. Chetan, H.K. Dakshina. Constraint method-based evolutionary algorithm (CMEA) for multiobjective optimization. In: E. Zitzler et al. (eds.), Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001), volume 1993 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2001, pp. 299–313.
[12] V. Srinivasan, G.L. Thompson, Algorithms for minimizing total cost, bottleneck time and bottleneck shipment in transportation problems, Naval Research Logistics Quarterly 23 (4) (1976) 567–598.
[13] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 257–271.