Proceedings of the 17th International Congress on Computer
Science Research CIICC´11
October 26-28, 2011, Morelia, Mexico

An Efficiently Computable Measure for Multiobjective Solution-front Quality

Elisa Schaeffer

Universidad Autónoma de Nuevo León (Mexico)
elisa.schaeffer@uanl.edu.mx

Abstract

In multiobjective optimization, the challenge is not to find one optimal solution but rather a set of solutions that all excel in terms of at least one of the objective functions and are tolerable in terms of the remaining objectives. In selecting such a set, among numerous options, it is necessary to apply a quality measure that determines whether or not the selection is good in some general sense. In this work, we propose and study one such quality measure, computable in asymptotic polynomial time in terms of both the number of solution candidates and the number of objective functions; existing literature often fails on this second point. We present some experimental results on the method.

Key words: Multiobjetive solutions, Pareto front, Quality front, measure.

1. Introduction

Optimization is, mathematically speaking, the process of choosing values for a set of variables subject to a set of restrictions in such a way that an *objective function* reaches an extremal value, that is, a maximum or a minimum, depending on the problem. Mathematically, maximizing a function $f()$ is equivalent to minimizing its negative $-f()$, for which in this paper we concentrate only on minimization. Special cases of optimization include those where the variables only take integer values (called *integer programming* [28]), those where all the restrictions are linear functions (called *linear programming* [2]), and convex optimization [4] (a class slightly larger than the previous), among others.

An assignment to the variables that satisfies all the restrictions is called a feasible solution. A feasible solution that reaches a minimum for an objective function is called an optimal solution, or simply the optimum. There may very well exist more than one optimal solution for a given optimization problem.

In multiobjective optimization (cf. [19, 22]), there are two or more (in practise, often much more than two) objective functions that usually contradict each other partially — a feasible solution that is an optimum to one function is not necessarily an optimum of the other function and vice versa. When there are numerous objective functions, this often gets quite messy from the point of view of choosing a feasible solution to be applied in practice, as the simple definition of an optimum simply does not extend to multiobjective optimization.

The concept that replaces the universal optimality of the single-objective case in multiobjective optimization is the *Pareto front*, which is a subset of feasible solutions that are Pareto optimal (also: Pareto efficient). The definition of Pareto optimality is based on a dominance relation: a feasible solution x Pareto-dominates another feasible solution x' if : $\forall i \in$ 1, 2 , 3, …, d : $f_i(x) \geq f_i(x')$, equation (1); meaning that x is always at least as good as x' under any objective function. The dominance is strict if x actually improves on at least one objective function. Based on these definitions, a feasible solution x is Pareto optimal within a set of feasible solutions X with respect to a set of d objective functions if x is not Pareto dominated by any other $x' \in X$. Figure 1 shows an example for a two-dimensional case (that is, $d = 2$) with $n = 10$ feasible solutions, $k = 4$ which are Pareto optimal. Note that the Pareto front spans by definition a convex hull to the set of feasible solutions. We will return to this later.

The subset formed by all Pareto-optimal solutions is called a Pareto front. A solution chosen in practice should in principle always be part of the Pareto front, as choosing any other feasible solution would imply that at least one of the objective functions could have been improved without harming the values of the other objective functions if one were to choose a solution that does form part of the front. An interesting, and in practise unpleasant observation is that when the number of objective functions is high, a very large percentage of the feasible solutions are Pareto-optimal [10, 14]. This makes in impractical to base algorithms on operating on the entire Pareto front and evidently makes it impossible for a human decision maker to review the options.

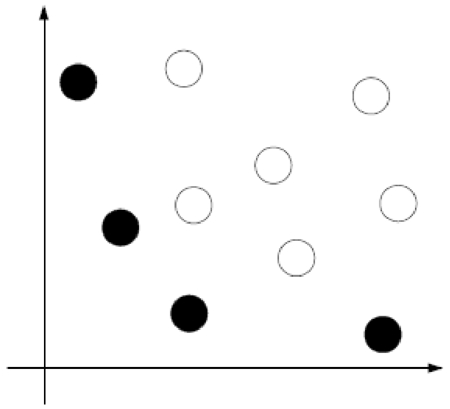

Figure 1. A set of feasible solutions, drawn as circles on a plane at the coordinates indicated by the values of the two objective functions (both being minimized). The black circles are Pareto optimal, as no other feasible solution dominates them.

The methods for creating a set of feasible solutions are numerous. As the exact solution of multiobjective optimization problems with several variables, restrictions, and objective functions is computationally very heavy (although integer programming as a decision problem belongs to the complexity class P, permitting its efficient solution, integer programming belongs to the class NP; all the existing algorithms scales very poorly in the worst case), most of the successful methods for generating good-quality feasible solutions are heuristics, meaning that there is no theoretical guarantee per sé of them producing the best possible solutions. Nevertheless, they do produce large quantities of solutions that are good enough for many practical applications. These methods include particle-swarm optimization [20], genetic algorithms [13, 16, 18], other evolutionary approaches [8], and methods that apply weights to convert the set of objective functions into a single linear combination of them [15]. Surveys on the field include those of Fonseca and Fleming [12], the dissertation of Van Veldhuizen [25] and the related article [26], as well as the book of Coello Coello et al. [6].

In this work, we do not attempt to generate either feasible solutions or Pareto fronts, but rather propose a method for comparing two Pareto fronts to each other in order to determine which is better in a sense that we will define in the next section. The method does not require, actually, that the compared subsets are in fact Pareto optimal — it is sometimes computationally hard to determine a Pareto front when the feasible solutions are numerous (as is the usual case in evolutionary algorithms) or when the objective functions are either tedious to evaluate or simply too many.

2. Front Quality

Given a large set of feasible solutions, the end user of an optimization method wishes to visualize somehow those that are Pareto optimal and then make, often by human judgement, a decision over which to select. The end user has an easier task if instead of presenting him or her with the entire Pareto front, a representative subset of it is shown, as this eases the cognitive load of comparing among the multiple alternatives by presenting less options and by making sure that the options shown are qualitatively different, that is, not very similar one to another.

This same situation occurs in an iterative heuristic algorithm. One iteration produced a possibly very large set of feasible solutions and the algorithm must choose a subset of these to be the basis of the next iteration. However, it is not a good choice to stick with only those solutions that are good with respect to a certain objective function or a subset of objective functions — the representatives should be somehow uniformly spanned over the Pareto front in the set of feasible solutions rather than clustered together. A third scenario is supposing that two or more algorithms for used to compute a Pareto front for the same problem. How to assess the quality of these solutions with respect to one another?

This is easiest to understand in the two-dimensional case, as it can be easily illustrated, but conceptually generalizes easily to any arbitrary higher dimension n. For simplicity, we restrict this work to objective functions that map feasible solutions, their set denoted by Ω, to real values $\Re$, that is, fi : $\Omega \rightarrow \Re$, that is, fi (x) = r. In our work, the number of variables that form the vector x is not relevant — we are only interested in the value-space of the objective-function vector (f1(), f2(), . . . , fd()). Each feasible solution is hence represented as a point in $\Re$d, that is, a d-dimensional vector v not of variable values, but of the values to which the d objective functions evaluate given x.

Now, for the end user choosing among options and well as for the iterative heuristic deciding which solutions to use for the next iteration and which to discard, a good subset of solutions is one that is either a subset of the Pareto front or a subset that is very close to the Pareto front (in the case that the evaluation of the Pareto-dominance relation is too costly, computation-wise). Also, a good subset is one in which the points are evenly separated in $\Re$d; this latter property is illustrated in Figure 2 for the two-dimensional case.

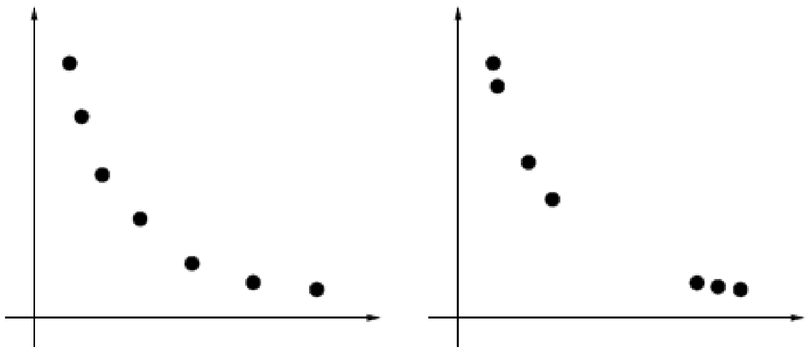

Figure 2. Both figures show a set of solutions in a two-dimensional space for a minimization problem. On the left, an evenly spaced front with good diversity is shown. On the right, there is an example of a front with poor diversity, as the solutions are grouped together. Notice that the theoretical best case is the origin, in the lower-left corner.

2.1 Existing work

As noted when defining a Pareto front in Section 1, the Pareto fronts can be computed by calculating the convex hull for the point set, limiting the candidate points by cutting with the hyperplane spanned by the minimum values of all *d* objective functions. This is due to the minimum point of each objective function necessarily being part of the Pareto front and the front itself being convex. A good algorithm for a few dimensions is the *quickhull* [1], with a randomized version for two dimensions [27]. These are efficient algorithms for two or three dimensions, or order $O(n \log n)$ where n is the number of points, but for higher orders, the mere input-processing steps suffer from the increasing number of dimensions, making the algorithm slow —the number of dimensions d appears as an exponent in the complexity. The good thing is that the outcome will necessarily be a Pareto front, but not a uniformly distributed subset, but the entire Pareto front. This would then need to be sampled or pruned in some sense to produce the uniformly spaced subset of a desired cardinality $k \leq n$.

Schott [23] manages uniformity using equidistant points on the axis in a sense protecting them on the front, whereas Meng et al. [29] propose a measure based on constructing a reference set — uniformly spaced — and the comparing the front candidates to that reference solution, which they limit to two dimensions and the unconstrained optimization. Zeng [30] also propose comparing the candidate fronts with the exact Pareto front, but extend their work for three or more dimensions. An established measure for solution subset quality is the hypervolume [17, 31] of the volume spanned by convex hull of the n points, computable in O(n log n+nd/2 log n) time for points in □d [3]. A survey of existing methods for multiobjective optimization solution quality assessment is given by Cheng et al. [5].

Deb and Jain [9] note that it is important for any metric to give values in a fixed range, such as (0, 1) for comparability and that slight improvements in the subsets should yield slight improvements in the metric, be computationally inexpensive, as well as scalable to an arbitrary number of objective functions. They also discuss separately the question of convergence towards the actual Pareto front and the diversity, that is, the ideally uniform separation of the front. In this work we propose a measure that can be used either as an exact measure or a randomized, approximate measure that requires less computation.

2.2 Proposed measure

As stated in previous sections, denoting the number of objective functions by d and assuming that each objective function maps an arbitrary structure that represents the solution candidate to a real number, the fitness of each solution is completely represented by a vector in $\Re^d$, a d-dimensional point. As Meng et al. [29], we assume as a starting point that the objective functions are normalized; in our work, we assume that the values are in (0, 1). We propose a measure that has polynomial asymptotic complexity in both the number of points n and the number of dimensions d.

In this section, for simplicity, we will use the term "front" to refer to an arbitrary subset of feasible solutions, whether or not it be a set in which there are no internally Pareto-dominated points. We denote the cardinality of this set by k. Usually, k « n, as the decision maker wishes to see just a few representative solutions and a small fraction of the solutions generated in one iteration will be used as a basis for the next one. Treating k as a constant parameter, it is easy to see that the computational complexity of the measure proposed in this work is controllable by the parameter k.

First, a distance measure is imposed on the set of d-dimensional points that form a front F; we denote the cardinality of the front as $|F| = k$. In this work we use the Euclidean distance for simplicity, that is,

$$d_{\mathbf{v},\mathbf{w}} = \sqrt{\sum_{i=1}^{d}(v_i - w_i)^2}, \qquad (2)$$

where v and w are vectors in $\Re^d$ and v_i is the i-th element of the vector v. In fact, any distance measure that fulfills the triangle inequality would do just as well. Here the important thing is that the distance computation for two points in $\Re^d$ takes $O(d)$ time.

The k-by-k matrix formed by the distances is denoted by D and is computable in $O(dn^2)$ time. For extremely high values of d, we leave for further work the possibility of computing the distance only for a small random sample of the dimensions at each step in order to bring down the computational cost; for the algorithm application scenario, this random sample could be retaken at each iteration in order to seek representation of all objective functions over time, whereas in the decision-maker scenario the measure could be repeated a constant c times using distinct random samples and still save in computational complexity when comparing to using all d objective functions. However, as we show later in the experiments, the exact computation of the measure remains feasible for rather high values of d as long as k remains of moderate size.

We then compute a minimum spanning tree (MST) on a weighted graph (for brevity, we do not enter to details of graph-theoretical terminology —we recommend the textbooks of Cormen et al. [7] on the algorithmic aspects and Diestel [11] for the mathematical aspects) using the points $v \in F$ as vertices and the matrix D determines the edge weights; this can be achieved in polynomial time in n using the classical algorithms of Prim-Jarnik or Kruskal very simply in $O(n^2)$ time, which can be further improved on a priority-queue or heap-based implementation, especially for sparse graphs (see [21] for a relative new MST algorithm).

We modify the MST algorithm slightly to simultaneously keep track of the degree of each vertex in the resulting spanning tree and denote these values by deg (v). Also, the maximum edge weight employed in the MST is to be stored and is here denoted by ρ.

Now we borrow a concept from fractal geometry where sets of points are covered by boxes or circles[1] (see for example the work of) and compute the coverage overhead of the MST if a n-dimensional hyperball of diameter ρ /2 were to be placed at each point. For simplicity and efficiency, we ignore possible overlaps of hyperballs of vertices that are not neighbors in the MST. By definition of ρ, the longest edge will be covered exactly once. We compute sum of the fragments of each edge that are covered by both endpoints, which simplifies into computing the difference between ρ and the length of the edge:

$$\varepsilon(\mathbf{v}, \mathbf{w}) = \rho - d(\mathbf{v}, \mathbf{w}) \tag{3}$$

and write

$$\varepsilon(\mathcal{T}) = 1 - \frac{1}{k-1} \sum_{(\mathbf{v},\mathbf{w}) \in \mathcal{T}} \varepsilon(\mathbf{v}, \mathbf{w}) \tag{4}$$

for the global coverage overhead, smaller the better , normalized over the $k - 1$ edges of the MST. See Figure 3 for an illustration.

Should the MST be a line, as is the case in the example shown in Figure 3, this would be sufficient for proceeding to the next step of the measure computation. However, in general, sets of solutions in the objective space that need to be evaluated during the execution of an optimization algorithm are composed of points clustered together or points that are in a sense outliers to the rest of the front. Such structures easily rise to branching in the resulting MST.

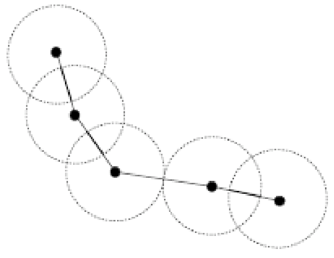

Figure 3. The redundancy in coverage of the front formed by connecting points into a minimum spanning tree, illustrated in a two-dimensional case. The covering ρ /2-radius circles are drawn with dashed lines, whereas the parts of the edges of the MST that are multiply covered by the circles are drawn thicker.

[1] The relation between the numbers of covering elements versus the size of each covering element determines some of the existing measures of fractal dimension.

In terms of what intuitively is front quality, any front candidate that branches is inferior to one that does not, as illustrated in Figure 4. The branching is easiest measures in terms of the vertex degrees: the higher the degree, the more severe the disadvantage. If our task was the optimization of the subset, we could attempt to cut off the branches or "straighten them out" by "jumping ahead", as in the well-known factor 2 approximation algorithm for the TSP, but as we merely wish to measure the quality of the front, simple penalization suffices for our purposes.

We therefore seek to impose a penalty to the redundant coverage of each edge in terms of the degrees of its endpoints. Note that this penalty cannot simply be a linear function of the degrees, as the sum of the degrees is always $2(k-1)$ for any MST. We use the following:

$$\xi(\mathrm{v}) = \begin{cases} \deg(\mathrm{v}) & \text{if } \deg(\mathrm{v}) > 2, \\ 0, & \text{if } \deg(\mathrm{v}) \leq 2, \end{cases} \tag{5}$$

as a very simple penalty function for degrees above two. We now write

$$\Xi(\mathcal{T}) = \sum_{\mathrm{v}} \xi(\mathrm{v}), \tag{6}$$

in order to normalize the penalty in terms of the ideal case, which is a line-like MST $\mathcal{T}^{+}$, producing zero total penalty, and what we consider the worst case, which is a star-topology MST $\mathcal{T}^{-}$ where one vertex is connected to all the rest, producing a total penalty $k-1$:

$$\xi(\mathcal{T}) = 1 - \frac{\Xi(\mathcal{T}) - \Xi(\mathcal{T}^{+})}{\Xi(\mathcal{T}^{-}) - \Xi(\mathcal{T}+)}, \tag{7}$$

for the global normalized degree quality, the larger the better. We write it in this fashion to be generalizable for other penalty functions that do not necessarily assign zero value to the ideal case, not the maximum value to the worst case; in this latter case one needs to additionally cut off the value of $\Xi(\mathcal{T})$ so it never exceeds $\Xi(\mathcal{T}^{-})$.

Also, a convex front is generally preferable to a non-convex one, for which the convexity must be somehow transformed into a measure over the MST created. We use the angles between consecutive edges in the MST, as illustrated in Figure 5. For simplicity and efficiency, we compute for each vertex of degree at least two the angle between its two *longest edges* (in case of branching) and denote this angle, when defined, by α_{v}. We then compute the average μ_{α} over the defined angles and calculate for each vertex a measure of deviation from the average, normalized to (0, 1):

$$\alpha(\mathrm{v}) = \begin{cases} |\mu_{\alpha} - \alpha_{\mathrm{v}}|, & \text{if } \deg(\mathrm{v}) \geq 2, \\ 0, & \text{if } \deg(\mathrm{v}) < 2, \end{cases} \tag{8}$$

Note that neither is a Pareto front; on the right, the first three points from the top and the last two points from the bottom along the line would not form part of a Pareto front which will only give zero if all the points lie on the same exact line in $\mathfrak{R}^{d}$ and small values if the curvature is smooth and line-like.

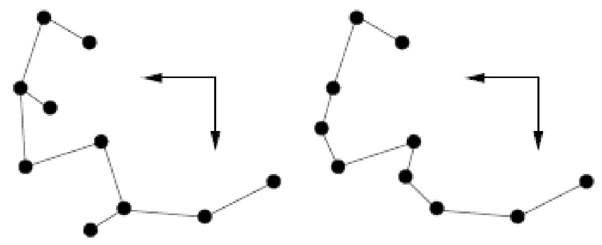
Figure 4. On the left, a branching MST, that is, a tree that has vertices with degree greater than two. On the right, an MST with no branching that should intuitively correspond to a higher-quality front. The arrows indicate the direction of the optimum for each objective function. The degrees of the vertices are used to measure the branching factor of an MST.

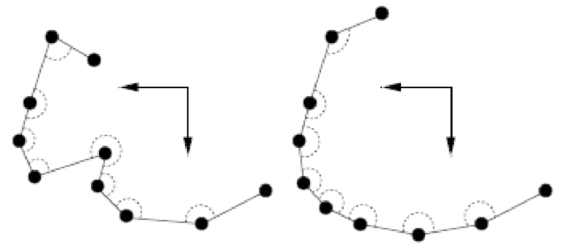
Figure 5. On the left, a non-convex MST, and on the right, a convex MST that should intuitively correspond to a higher-quality front. The arrows indicate the direction of the optimum for each objective function. The angles used to measure convexity are indicated by dashed lines.

Hence we write

$$\alpha(\mathcal{T}) = 1 - \frac{1}{2\pi k} \sum_{\mathrm{v} \in \mathcal{F}} \alpha(\mathrm{v}), \tag{9}$$

to account for the angles produced by the edge pairs considered. Another option for smoothness would be the autocorrelations of the very degrees along the MST, which we leave to further work.

This measure does not yet take into account the *orientation* of the front in the space $\Re^d$ —we use the angles formed between the origin and two consecutive vertices (that is, neighbors in the MST), computed simply as

$$\vartheta(\mathrm{v}, \mathrm{w}) = \arccos \frac{\mathrm{v} \cdot \mathrm{w}}{|\mathrm{v}|\,|\mathrm{w}|}. \tag{10}$$

A front is oriented facing the origin if these angles are large and perpendicular to that if they are generally very small. We hence take the sum of these angles, normalized by $k - 1$ as there are $k - 1$ edges in the tree, and include it in the measure:

$$\vartheta(\mathcal{T}) = \frac{1}{k-1} \sum_{(\mathrm{v},\mathrm{w}) \in \mathcal{T})} \vartheta(\mathrm{v}, \mathrm{w}), \tag{11}$$

the bigger the better.

We still have the distance from the optimum to account for. If we sum the distances from the origin (the theoretical ideal point, assuming normalization of the objective functions to a non-negative range) to all points and divide by the square of *k* to normalize,

$$\mathcal{D}(\mathrm{v}) = \sqrt{\sum_{i=1}^{d} (v_i)^2}, \tag{12}$$

and again we normalize and revert this to be a penalty:

$$\mathcal{D}(\mathcal{F}) = 1 - \frac{1}{k \cdot d} \sum_{\mathrm{v} \in \mathcal{F}} \mathcal{D}(\mathrm{v}), \tag{13}$$

the normalization being this way as we sum k distances, each of which at most d, given that the objective functions are normalized to (0, 1). The value of *D*(*F*) will be smaller to fronts that are in general oriented closer to the origin and can be used as a divisor to increase further those measures that correspond to good fronts. The normalization here assumes that the values $v_i \in (0, 1)$ – if this is not the case, values above one may result, although the measure is still comparable between fronts of different sizes for the same problem instance.

We then proceed to calculate the front quality *Q*(*F*) by combining the defined penalties in the following simple manner:

$$\mathcal{Q}(\mathcal{F}) = \tfrac{1}{5}\Big(\vartheta(\mathcal{T}) + \mathcal{D}(\mathcal{F}) + \varepsilon(\mathcal{T}) + \alpha(\mathcal{T}) + \xi(\mathcal{T})\Big) \tag{14}$$

where *T* is the set of edges included in the MST and the components as defined earlier in this section in Equations 11, 13, 4, and 9, and 7. Do note that in face were are facing a multiobjective optimization problem: the ideal front would minimize all penalties.

The measure will give values in (0, 1) and we write it as a percentage, that is, 100×*Q*(*F*)%. Higher values of the measure indicate better fronts. Note that this is but one possible combination (we also experimented with the product, but the values are very small and less readable) and that it could easily be parametrized; we leave to future work the inclusion of weights to the penalty functions, together with mechanisms for automatic and adaptive parametrization.

3. Experimental Results

In this section, we demonstrate the behavior, scalability, and applicability of the proposed measure in different settings. We begin by offering some more visual examples in low dimensions, then vary both the number of solutions and the number of objective functions to study the scalability aspects, and finally integrate the proposed measure to a simple evolutionary algorithm to demonstrate a possible application.

3.1 Visual examples

We begin by demonstrating the behavior of the proposed measure on small two-dimensional examples that are easy to visualize and understand. Figure 6 shows six point sets of slightly distinct cardinalities and.the corresponding MST constructions and the values obtained for the proposed quality measure. We observe that the measure prefers smooth and evenly spaced curves that are oriented as a wave towards the origin (the right-most column), less so vertical arrangements (middle column), even less non-linear arrangements (lower left corner), and the least clustered arrangements (upper left corner).

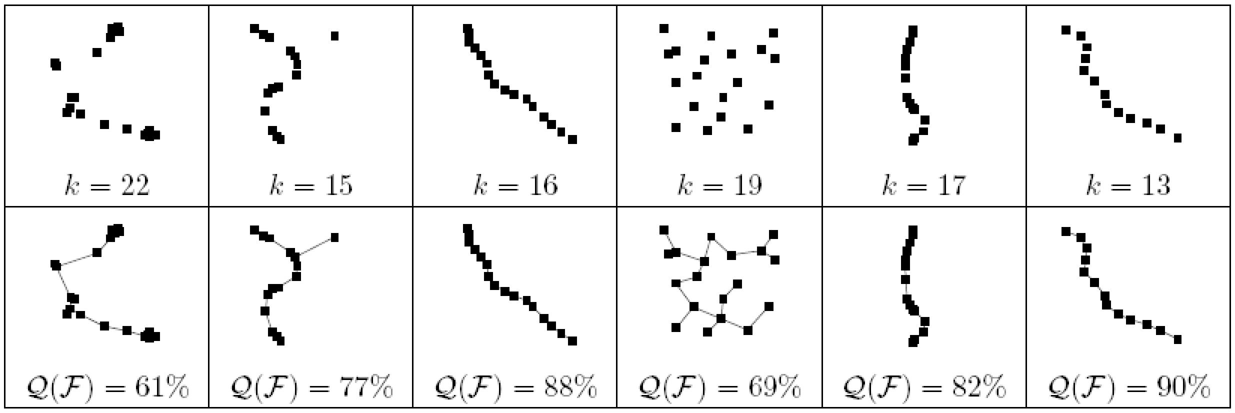

Figure 6. Above, six example point sets in two dimensions; the number of points is shown for each set. Below, their resulting MST together with the quality-measure value of each.

To illustrate the manner in which the measure changes when the front is modified, we include in Figure 7 a few examples on how the quality measure changes for similar fronts where the major difference is in the presence or absence of a certain point or a few points. It can be seen that the proposed measure rewards even placement and smoothness of the front. It is also visible that the measure evolves with small changes as small modifications are introduced to the front as recommended by Deb and Jain [9].

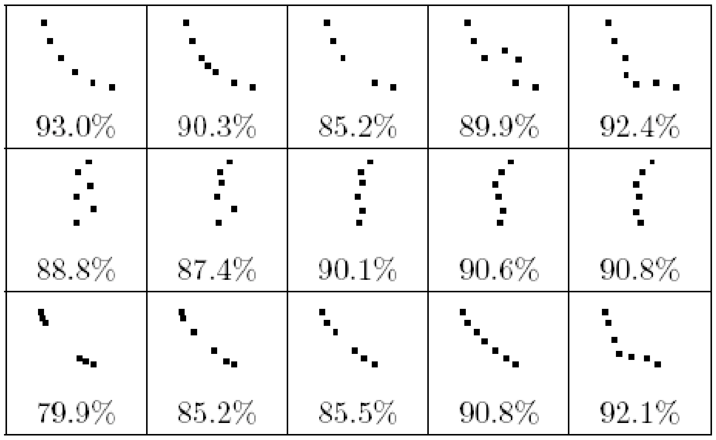

Figure 7. Examples of similar fronts (one group per line) and the corresponding values of the proposed quality measure *Q*(*F*) for each front.

3.2 Scalability

We generated instances by placing *n* points uniformly at random in a *d*-dimensional hypercube (we generated a uniform pseudo-random angle β and the used $f_i = (sin\ (i \cdot \beta) + 0.5)$ for i = 1, 2, ..., d. This is obviously artificial, but the purpose is to study the scalability of the proposed measure in terms of the runtime when varying both n and d. We chose n = 16, 32, 64, . . . and d = 2, 4, 8, 16, . . . for the experiments and ran 30 independent trials for each combination, registering the runtime in seconds per each and then computing the average and standard deviation over the set of repetition. All experiments were ran, non-exclusively on the background of normal use, on a MacBook Air with a 1.4 GHz Intel Core 2 Duo with 4 GB of 1,067MHz DDR3 RAM under OS X 10.7 Lion. The scalability experiment program code was written in ANSI C.

Figure 8 shows how the values vary for some fixed d and multiple n and vice versa. Not all combinations were included, as the curves overlap, which affects adversely the legibility of the graphic. We can see that the runtime increases clearly faster in terms of the number of points than with the number of objective functions. This is good, as it permits us to continue to higher dimensions with little complication, as long as we maintain k small, which is not at all impossible in many practical applications of these types of measures.

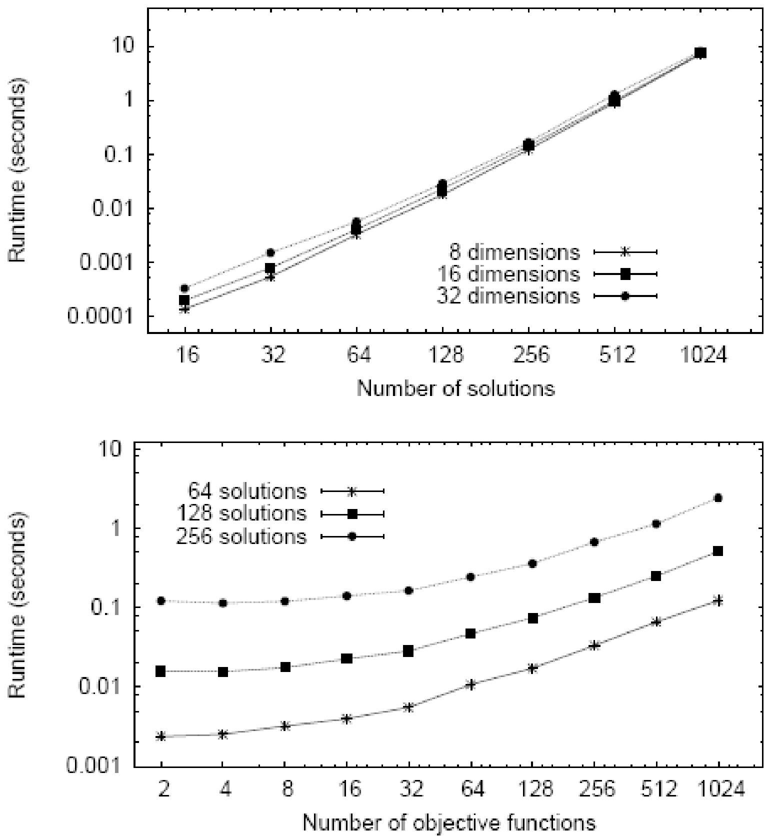

Figure 8. Above, the average runtime (in seconds) for different values of n with $d \in \{8, 16, 32\}$, and below, the same for d with $n \in \{64, 128, 256\}$; the standard deviation is drawn as an error bar on each dot, but these are too small to be seen. Notice the logarithmic scale of the axes.

The relation between d and n and the runtime is illustrated in Figure 9, where it is clearly visible that the proposed method is not so much sensitive to the number of objective functions as it is to the number of points; for the decision-makes scenario where only a few points are to be selected for consideration, this is very practical. For large, low-dimensional sets, the existing methods in the literature outperform the proposed approach, whereas for higher dimensions, this measure is computationally competitive.

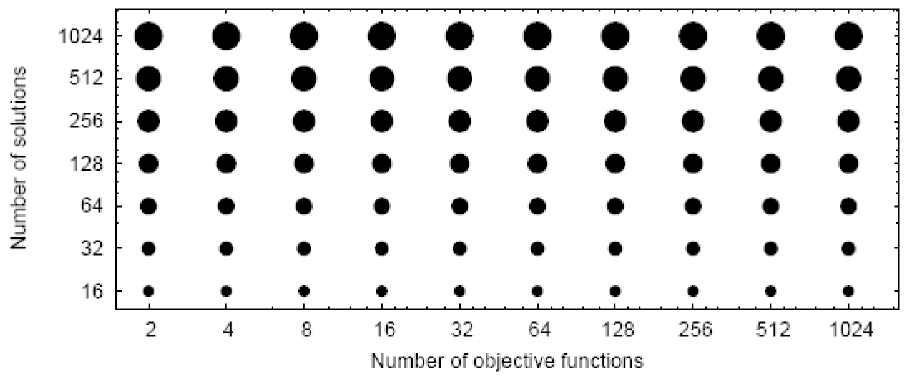

Figure 9. The size of each circle indicates the logarithm of the average runtime for each (n, d) pair. Notice also the logarithmic scale on both axes.

3.3 Graph coloring

As generating data uniformly at random is rarely realistic, we also experimented on a combinatorial optimization problem, namely the *graph coloring* optimization problem. The corresponding decision problem asks whether it is possible to color a given graph $G = (V,E)$ using no more than γ colors such that no pair of vertices from V that is connected by an edge in E shares a color; this is known to be NP complete. Consequently, as verifying the correctness of a suggested coloring is doable in polynomial time and logarithmic space, the optimization problem is NP hard.

A *color assignment* $\boldsymbol{C}$ is a listing of the colors assigned to each vertex in V. A color class $C_i \subseteq \boldsymbol{C}$ is a set of vertices all colored with the color i; we are here interested in the cardinalities (that is, the number of elements in each) of the color classes, as we will work with the following multiobjective formulation of the problem: given a graph $G = (V,E)$, find a color assignment $\boldsymbol{C}$ that minimizes

1. $f_1(\boldsymbol{C})$ = the number of colors γ used in the coloring divided by $|V|$,
2. $f_2(G, \boldsymbol{C})$ = the number of conflicts present divided by $|E|$,
3. $f_3(G, \boldsymbol{C})$ = the number of triplets in G where the two end points share color, divided by $\binom{|V|}{3}$
4. $f_4(\boldsymbol{C})$ = the minimum of the maximum of $|C_i|$ divided by $|V|$,
5. $f_5(\boldsymbol{C})$ = the maximum of the minimum of $|C_i|$ divided by $|V|$,
6. $f_6(\boldsymbol{C})$ = the maximum difference over all i from the average of the $|C_i|$ divided by $|V|$,

with

$$f_3(G,\mathcal{C}) = \Big|\{\{u,v,w\} \mid u,v,w \in V, (u,v),(v,w) \in E \text{ s.t. } u,w \in C_i\}\Big| \tag{15}$$

counting effectively second-neighbor color conflicts, and

$$f_6(\mathcal{C}) = \max_{i \in [1,\gamma]} \left| |C_i| - \frac{1}{\gamma} \sum_{j=1}^{\gamma} |C_j| \right|. \quad (16)$$

The divisions by graph order and size are included in order to normalize the objective functions to yield comparable values for graphs that differ greatly in order and/or size.

We implemented in Python a very simple genetic algorithm, designed to avoid parameter selection, using $\mathcal{C}$ represented as a fixed-order vector, one element per each vertex, as the chromosome and performing linear crossovers at a uniform random point, picking each pair of distinct solutions once to cross in each iteration, as well as applying a uniform random mutation in one element to each solution present, interchanging the color of one vertex with that of another vertex within that coloring. Duplicate solutions are not permitted to enter the population and all colorings are simplified by relabeling the colors with consecutive values 1, 2, . . . when intermediate colors are absent. Note that by design there is no mechanism whatsoever to explicitly and purposefully improve the population at the expansion phase —all solutions are equally mutated and crossed.

The number of solutions per iteration was fixed at $h = \lceil \log(|V|) \rceil + d$, where $d = 6$ is the number of objective functions. For the number of generations to compute we used simply $|V| + h$, which would be very little for practical purposes, but serves well for our demonstration. The initial solutions for the first iteration are generated by first choosing uniformly at random a value $\gamma \in [1, |V|]$ and the placing a color from this range for each vertex, also uniformly at random.

We select the set of h solutions to pass on to the next generation. In the first phase, we use a greedy fill that incorporated solutions incrementally in random order by greedily optimizing of the resulting front (by a subprocess call to the ANSI C routine implemented for the previous experiments): a candidate for inclusion to the chosen set of is accepted when it either improves the existing front, yields at last 99% of the quality of the front formed by the previously selected solutions[2], or simply attains a quality at least 90% when included in the set. The candidate is rejected when it fails to fulfill all three conditions. This growth mechanism is cut off when cardinality h is reached.

On a second pass, carried out if h solutions have not yet been selected, each presently not selected solution is reconsidered for a random preferential inclusion: each candidate v is accepted with probability to give preference to including those that have lower values for the objectives on average, and again the procedure is cut off when cardinality h is reached.

$$p_{\text{pref}}(\mathrm{v}) = 1 - \frac{1}{d} \sum_{i=1}^{d} f_i(\mathrm{v}) \quad (17)$$

2 Note that this percentage together with h limit how much the quality can decrease in total during the greedy fill phase, and hence should be adjusted to a tighter value when h is large.

On a third pass, should cardinality h still not be attained, presently unselected solutions are incorporated uniformly at random to reach the desired cardinality. We record the percentage of each type if fill (that is, the proportion of solutions that were chosen based on front quality, preferentially based on the objective functions, or uniformly at random) as well as the quality of the front with which the greedy procedure terminated at each iteration.

Note that in this simplistic scenario, we actually only evaluate the objective function upon considering a coloring for inclusion to the next-iteration seed population. Not all solution candidates necessarily get evaluated. We are not so much trying to solve the problem as we are seeking to show how the measure works.

For purposes of comparison, we executed this algorithm both with and without the greedy phase, 10 independent repetitions each way. We refer to the complete version as the greedy version and to the reduced version as the random version. We experimented on DIMACS format graph-coloring benchmark instances [24]; we show here the results for one graph only, for brevity —combining results for different-structured graphs would result in uninformative or even confusing plots.

We chose a Mycielski transformation graph myciel4.col, which is small enough to visualize comfortably the resulting fronts with $|V| = 23$ and $|E| = 71$. We report the averages and standard deviations over the 10 replicas. In Figure 10, we plot the front quality $Q(F)$ of the population at each iteration (both for the greedy and the random version) and the fill percentage for the greedy version (the third pass was not needed in any of the replicas for either version, for which it is not plotted), together with the minimum and average value per objective function per iteration of each f_i.

Figure 10 shows that the greedy procedure almost always fills the selected solution set and that this mechanisms maintains a higher front quality than the random preferential method based purely on Equation 17.

We also observe that the front qualities are relatively stable over the 10 replicas, whereas the fill percentage of the greedy step varies greatly, as is natural to a greedy search as it occasionally gets stuck at a local optima from which it is difficult to break out. It is also noteworthy that some of the averages remain well above the minimums, indicating an adequate diversity within each individual objective function.

To examine whether the greedy method actually helped in the front diversity, we examine the populations produced at the last iteration of each run for both of the methods. As visualizing the six-dimensional space is not straightforward, we plot a few projections to two-dimensional space: f1 versus f2, f3 versus f4, and f5 versus f6, these being but a sample of the $\binom{6}{2} = 15$ possible projections, for the sake of brevity. We plot the overlays of the 10 resulting fronts, each with $|V|$ solutions, of the independent replicas, all jointly, to illustrate the shape of the resulting front. The results are shown in Figure 11.

We can see in Figure 11 that the overlayed fronts of the greedy procedure (top row) have in a sense a similar shape —a cloud-like form on the leftmost column, a line-like form on the middle column, and a vertical pillar on the right-hand side —but that in all cases the solutions provided by the greedy procedure are on average reaching the same minimums than the random procedure, but maintaining a wider front.

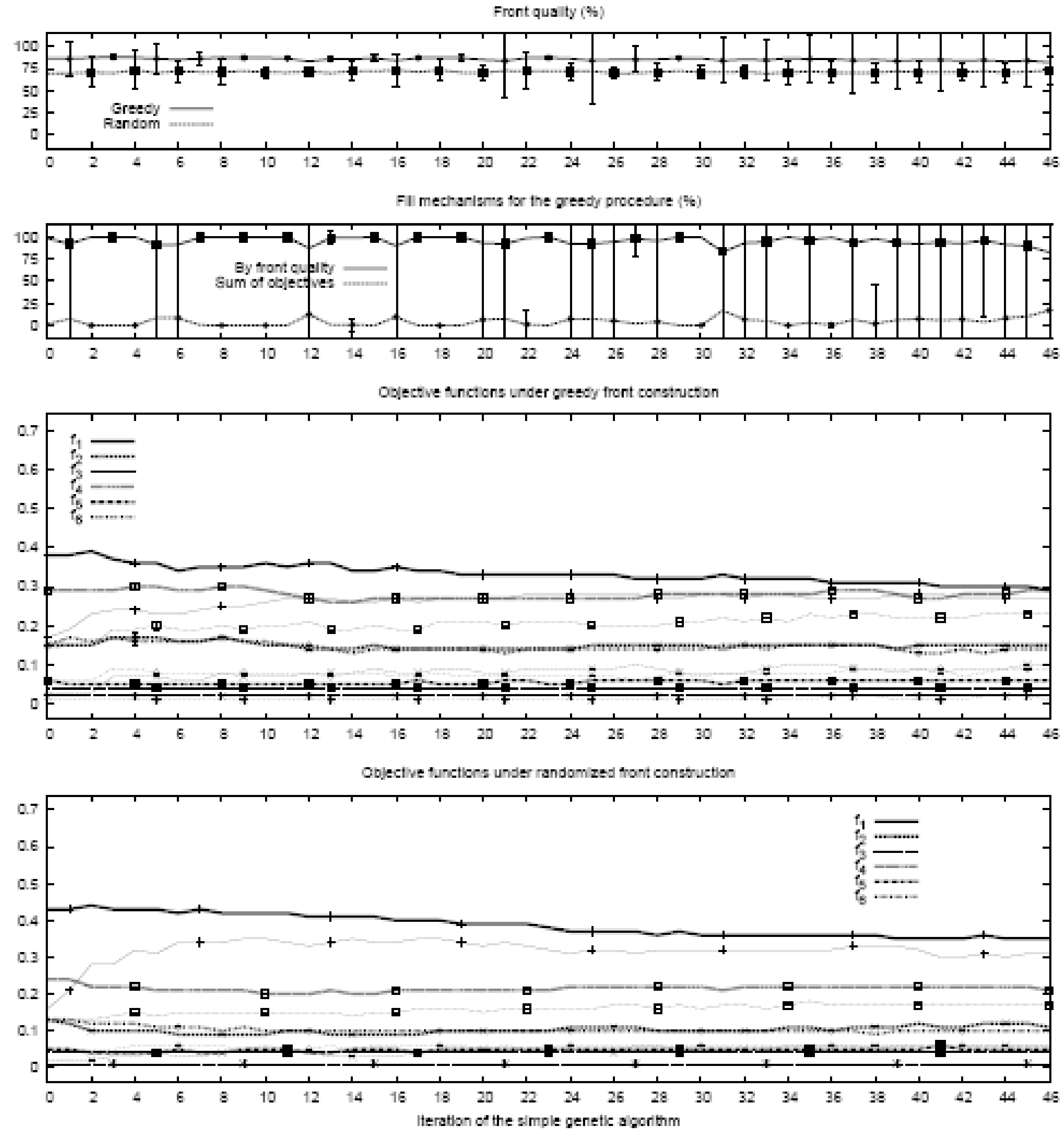

Figure 10: The Mycielski transformation graph. From the top: the front quality, the fill percentage of the greedy procedure, the performance with the greedy procedure, and without it; the thick lines show the average value of each objective function and the thin lines of the same style indicate the respective minimum value.

4. Conclusions and Future Woks

We have presented a novel quality measure for comparing solution subsets in multiobjective optimization, specially suited for selecting a subset to present to a decision maker or for incorporation in an iterative population-based optimization method that needs to prune out a population from those examined in the previous iteration to be considered for the next iteration, such as evolutionary algorithms. We have argued formally and demonstrated by experiments that the proposed measure is efficiently computable, especially in terms of the number of objective functions, and well-suited for evaluating multiple small subsets of very high dimensionality. We have presented visualizations and experimental results to explain the function and the properties of the proposed measure.

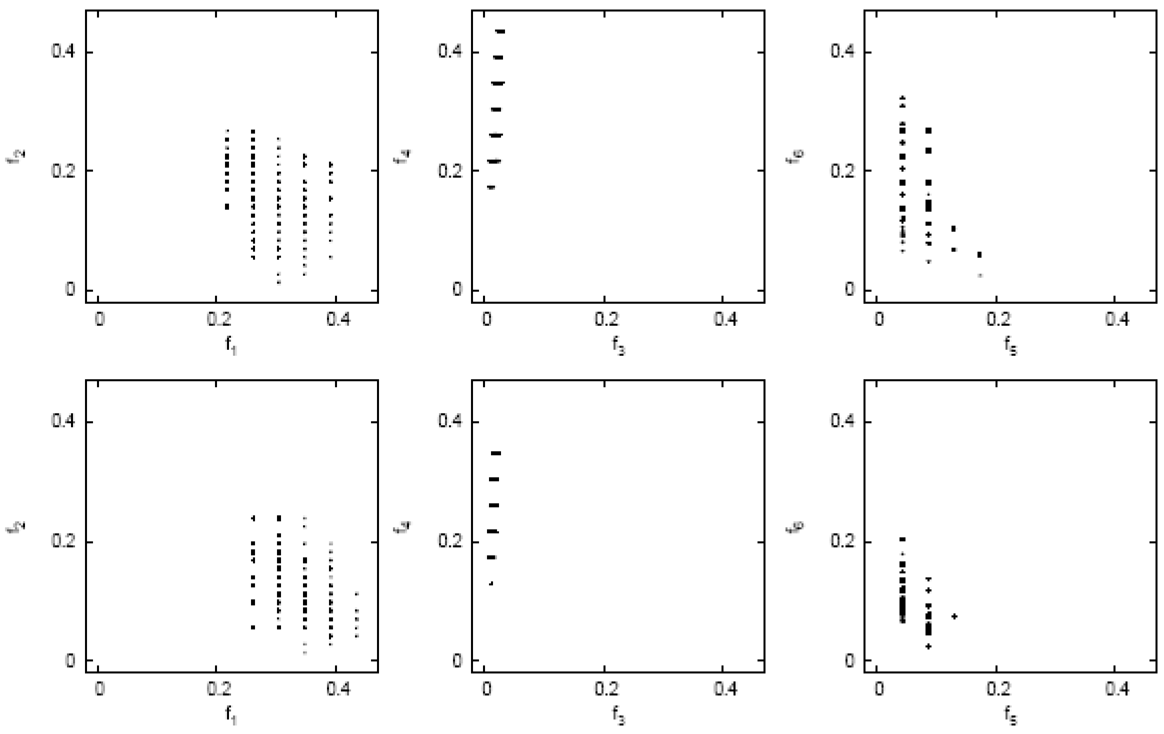

Figure 11: Overlays of 10 projected final fronts for the Mycielski transformation graph. On the top row, the fronts produced by using the greedy procedure, and on the bottom row, those resulting from the random fill.

In future work, we hope to study more combinations of the components of the suggested measure, as well as adaptive parameter adjustment strategies for including problem-specific weights. We also plan to experiment on randomized versions of the measure for extremely high dimensions and for very large solution sets.

References

[1] C. B. Barber, D. P. Dobkin, and H. Huhdanpää. The quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software (TOMS), 22(4), 1996.

[2] D. Bertsimas and J. N. Tsitsiklis. Introduction to Linear Optimization. Athena Scientific, Nashua, NH, USA, 1997.

[3] N. Beume. S-metric calculation by considering dominated hypervolume as Klee's measure problem. Evolutionary Computation, 17(4):477–492, 2009.

[4] S. Boyd and L. Vanderberghe. Convex optimization. Cambridge University Press, Cambridge, UK, 2004.

[5] P. Cheng, J.-S. Pan, L. Li, Y. Tang, and C. Huang. A survey of performance assessment for multiobjective optimizers. In Proceeding of the 2010 Fourth International Conference on Genetic and Evolutionary Computing, Washington, DC, USA, 2010. IEEE Computer Society.

[6] C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen. Evolutionary algorithms for solving multiobjective problems. Genetic and evolutionary computation. Springer Science+Business Media, 2007.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. McGraw-Hill Book Co., Boston, MA, USA, second edition, 2001.

[8] D.W. Corne, J. D. Knowles, andM. J. Oates. The Pareto Envelope-Based Selection Algorithm for Multiobjective Optimization, volume 1917 of Lecture Notes in Computer Science, pages 839–848. Springer-Verlag, Berlin/Heidelberg, Germany, 2000.

[9] K. Deb and S. Jain. Running performance metrics for evolutionary multi-objective optimization. Technical Report Kangal Report No. 2002004, Kanpur Genetic Algorithms Laboratory, Indian Institute of Technology Kanpur, Kanpur, India, 2002.

[10] K. Deb, Pawan, and A. Jain. Distributed computing of pareto-optimal solutions with evolutionary algorithms. In Proceedings of the second international conference on Evolutionary multi-criterion optimization, Berlin/Heidelberg, Germany, 2003. Springer-Verlag.

[11] R. Diestel. Graph Theory, volume 173 of Graduate Texts in Mathematics. Springer-Verlag, New York, NY, USA, 2000.

[12] C.M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. Evolutionary Computation, 3(1):1–16, 1995.

[13] J. Horn, N. Nafpliotis, and D. Goldberg. A niched Pareto genetic algorithm for multiobjective optimization. In Proceedings of the First IEEE Conference on Evolutionary Computation; IEEE World Congress on Computational Intelligence, pages 82–87. IEEE, 1994.

[14] G. K. Kao and S. H. Jacobson. Finding preferred subsets of pareto optimal solutions. Computational optimization and applications, 40(1):73–95, 2008.

[15] I. Kim and O. de Weck. Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. Structural and multidisciplinary optimization, 29(2):149–158, 2005.

[16] R. Kumar and P. Rockett. Improved sampling of the Pareto-front in multiobjective genetic optimizations by steady-state evolution: A Pareto converging genetic algorithm. Evolutionary Computation, 10(3):283–314, 2002.

[17] M. Laumanns, G. Rudolph, and H.-P. Schwefel. Approximating the Pareto set: Concepts, diversity issues, and performance assessment. Technical Report CI-72/99, University of Dortmund, Collaborative Research Centre 531 Computational Intelligence, Dortmund, Germany, 1999.

[18] S.-Y. Liong, S.-T. Khu, and W.-T. Chan. Derivation of pareto front with genetic algorithm and neural network. Journal of Hydrologic Engineering, 6(1):52–61, 2001.

[19] K. Miettinen. Nonlinear multiobjective optimization. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[20] K. Parsopoulos and M. Vrahatis. Particle swarm optimization method in multiobjective problems. In Proceedings of the 2002 ACM symposium on Applied computing, pages 603–607, New York, NY, USA, 2002. ACM.

[21] S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. Journal of the ACM, 49(1), 2002.

[22] Y. Sawaragi, H. Nakayama, and T. Tanino. Theory of multiobjective optimization. Academic Press, Orlando, FL, USA, 1985.

[23] J. R. Schott. Fault tolerant design using single and multicriteria genetic algorithm optimization. PhD thesis, Draper Laboratory, Massachussetts Institute of Technology, Cambridge, MA, USA, 1995. Master's thesis.

[24] M. Trick. Graph coloring instances, 1996. http://mat.gsia.cmu.edu/COLOR/ instances.html.

[25] D. A. Van Veldhuizen. Multiobjective evolutionary algorithms: classiVcations, analyses, and new innovations . PhD thesis, Air Force Institute of Technology, Wright Patterson AFB, OH, USA, 1999.

[26] D. A. Van Veldhuizen and G. B. Lamont. Multiobjective evolutionary algorithms: analyzing the state-of-the-art. Evolutionary computation, 8(2):125–147, 2000.

[27] R. Wenger. Randomized quickhull. Algorithmica, 17(3):322–329, 1997.

[28] L. A. Wolsey. Integer Programming. Wiley-Interscience, 1998.

[29] H. yunMeng, X. hua Zhang, and S. yang Liu. New quality measures for multiobjective programming. In Advances in natural computation, volume 3611 of Lecture Notes in Computer Science, pages 1044–1048. Springer-Verlag, Berlin/Heidelberg, Germany, 2005.

[30] S. Zeng, G. Chen, R. Wang, H. Li, H. Shi, L. Ding, and L. Kang. A new technique for assessing the diversity of close-pareto-optimal front. In Proceedings of the IEEE World Congress on Computational Intelligence, pages 344–349, 2008.

[31] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms — a comparative case study. In Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature, pages 292–304, London, UK, 1998. Springer-Verlag.