

# Simple Seeding of Evolutionary Algorithms for Hard Multiobjective Minimization Problems

Klaus Truemper

Department of Computer Science,  
University of Texas at Dallas,  
Richardson, TX 75083, U.S.A.

**Abstract.** The paper describes a simple seeding method to be used with evolutionary algorithms for the solution of hard multiobjective minimization problems. Included are computational results for a specific implementation.

**Key words:** evolutionary algorithm, multiobjective optimization, seeding, hybrid method

## 1 Introduction

Evolutionary algorithms (EAs) have proved to be effective tools for the multiobjective minimization problem MOP, which has the following form [3].

$$\begin{aligned} \min \quad & \{f_1(x), f_2(x), \dots, f_t(x)\} \\ \text{s. t.} \quad & g_i(x) \leq 0; \quad i = 1, 2, \dots, m \\ & l \leq x \leq u \end{aligned}$$

where the objective functions  $f_k : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $k = 1, 2, \dots, t$ , are in conflict. The bound vectors  $l$  and  $u$  for  $x$  are finite. The objective and constraint functions are either given explicitly, or objective and constraint values are obtained by querying a black box.

A *decision vector*  $x$  of MOP produces the *objective vector*  $(f_1(x), f_2(x), \dots, f_t(x))$ . A *feasible* decision vector  $x$  satisfies  $g_i(x) \leq 0$ , for all  $i$ , and  $l \leq x \leq u$ . Such a vector is *Pareto optimal* if one cannot improve uniformly upon the corresponding objective vector in the following sense: There is no feasible vector  $y$  such that, for all  $k$ ,  $f_k(y) \leq f_k(x)$ , and strict inequality holds for at least one  $k$ . The set of objective vectors produced by the set of Pareto optimal decision vectors is the *Pareto front*. We want a reasonably precise, discrete representation of the Pareto front.

We consider here MOP instances that are potentially hard in the following sense.

- (1.1) Minimization of each objective function, by itself, over the feasible region may already be a difficult problem; in particular, each such function may have a number of local minima.

- (1.2) The objective functions and constraint functions may not be smooth.
- (1.3) The feasible region may not be a convex set.
- (1.4) If a black box supplies function values, then it may be expensive in the sense that the time for evaluating one decision vector is large; say, run time for one evaluation is one second or more, and may be as high as several minutes.
- (1.5) The problem may have to be solved from a cold start in the sense that there is no prior information in the form of decision vectors or objective vectors.

For hard MOPs, various *hybrid algorithms* have been proposed where EAs and other methods are combined; see the survey of [12]. In this paper, we describe one such hybrid algorithm and computational results for an implementation. In the terminology of [12], the hybrid algorithm considered here is an intelligent paradigm followed by an EA. The intelligent paradigm is *seeding*, which creates a starting population for the EA that isn't just a random selection. The seeding employed here consists of derivation and insertion of a select few decision vectors that produce objective vectors of the Pareto front. This seemingly elementary approach, when appropriately implemented, turns out to be surprisingly effective for the solution of hard MOPs.

The paper proceeds as follows. The next section, 2, reviews prior work. Section 3 motivates the seeding process, while Section 4 has implementation details. Section 5 has computational results, including solution of a hard Engineering problem.

## 2 Brief Review

It's impossible to do justice here to the vast body of algorithms and ideas that have been produced over the past sixty years for the computation of the Pareto front. An excellent recent book covering the various concepts and approaches is [3]. Here, we focus on EAs.

Grossly simplifying the situation, one could say that an EA for MOP starts with a sufficiently large population of solution vectors, then iteratively modifies that population using relatively simple rules until a population has been found that produces a reasonably precise representation of the Pareto front. Intuitively speaking, the iterative manipulation of large populations according to simple rules is both a strength and weakness of EAs for MOP. It is a strength since the algorithms evaluate in each iteration a relatively large portion of the feasible region to find better objective vectors, and thus have a good chance to avoid getting trapped by local minima. It is a weakness since the simple rules may not be able to cope with complex objective functions that can only be analyzed by detailed local investigation.

As a result, EAs for MOP are at a disadvantage when directly used for minimization of complicated single objective functions. Indeed, specialized single-function minimization techniques developed for the latter task have proved to be far superior to EAs for MOP. See, for example, the impressive results reported in extensive tests of derivative-free single-function minimization methods

in [17, 23], and contrast this with the difficulties encountered by the excellent EA NSGA-II for MOP when solving far simpler single-function minimization instances [7]. We emphasize that the above remarks do not apply to EAs especially designed for single-function minimization where small solution sets are used; see, for example, [18].

Since the Pareto front of any instance of MOP always contains, for each  $k = 1, 2, \dots, t$ , at least one objective vector where the component objective function  $f_k(\cdot)$  attains its minimum value, we expect that EAs for MOP have difficulty solving MOP instances where minimization of at least one  $f_k(\cdot)$  is hard.

### 3 Solution Approach

In attempts to eliminate the above mentioned disadvantage of EAs for MOP, hybrid algorithms have been proposed; see, for example, [11, 24], where serial-hybrid and concurrent-hybrid versions of the EA NSGA-II [6] are described. Here, the solution vectors of the EA are improved by local search either after the EA has stopped, which is the serial-hybrid approach, or during each iteration of that algorithm, which is the concurrent-hybrid method. For a number of other hybrid methods using various component schemes and approaches, see the discussion and references of [10, 12, 21, 25].

The serial-hybrid method of [11] is shown in [24] to be inferior to the concurrent-hybrid method of [24]. Nevertheless, we follow the lead of [11], but invert, so to speak, the former scheme. In the terminology of [12], we thus have an intelligent paradigm followed by an EA. The intelligent paradigm is *seeding*, which creates a starting population that isn't just randomly selected but has *seed vectors*, for short *seeds*, inserted. There are a number of implementations of that idea [12]. We skip details and mention only that we use seeding here by inserting into the initial population a few seeds that produce part of the Pareto front. This is in contrast with the typical approaches summarized in [12], where good but not necessarily optimal solutions are utilized. It also differs from the seeding used in [13], where an approximation of objective function gradients is used to derive an initial population whose objective vectors are close to the Pareto front. The latter method requires the objective functions to be differentiable in contrast to condition (1.2), and it does not handle general constraints of the form  $g_i(x) \leq 0$ ,  $i = 1, 2, \dots, m$  allowed here.

Since the proposed seeds produce objective vectors of the Pareto front, one may conjecture that the EA rather quickly discovers that improvement of those point is not possible, and also rapidly finds additional Pareto points in the neighborhood of these initial points. Arguing inductively, one may conjecture that the EA constructs the Pareto front by growing it, so to speak, recursively from the initial points. Section 5 includes an example of the conjectured process where NSGA-II is the EA. Indeed, when for that case the results of the iterations of NSGA-II are displayed graphically, the growth of the Pareto front after a few iterations looks like a crystallization process that starts from the initial objective vectors produced by the seeds.

In the typical hard MOP case, the above conjecture is too optimistic, and NSGA-II as EA does not produce such straightforward growth of the Pareto front from the seeds. Indeed, in experiments the following takes place in the vast majority of cases. The EA creates portions of the Pareto front near the seeds and then fills in more and more points of a surface that deceptively looks like the Pareto front. We call the construction of the surface the *crystallization phase* of the EA. But that surface is just an approximation of the Pareto front, and in subsequent iterations the EA deforms the surface gradually until it has determined the Pareto front in a sufficiently accurate manner. Two examples are discussed in Section 5.

We have chosen NSGA-II as EA for the experiments since it has exhibited remarkable performance for numerous applications; see, for example, [14, 16, 26]. It is a reasonable, as yet untested hypothesis that the experimentally established performance is also achieved when similarly effective EAs are used, for example, the algorithm SPEA2 [27].

The proposed use of seeds is different from the notion of *reference points* used in an extension of NSGA-II called NSGA-III [15]; see also the discussion of predecessor versions of NSGA-III [5, 4]. The reference points are distributed in the search space and assure diversity of solutions.

How can the above idea be implemented? The most straightforward way would be solving the single-function minimization problem for each objective function by an effective method and declaring the resulting optimal solutions to be the seeds for application of the EA to the MOP.

This appealing approach requires careful implementation. Specifically, the vectors computed by single-function minimization not necessarily produce points of the Pareto front, since in each case just one function is minimized. But we do achieve that effect when we use a weighted sum of the objective functions where the single objective function to be minimized has a large positive weight, and the other objective functions have appropriate smaller positive weights [22]. This rule works well when the MOP has  $t = 2$  objective functions. But it may fail badly when  $t \geq 3$ , in the sense that far less than  $t$  distinct seeds are produced. An example is the MOP

$$\begin{aligned} \min \quad & \{x_1, x_2, \dots, x_t\} \\ \text{s. t.} \quad & \sum_{j=1}^t x_j^2 \geq 1 \\ & 0 \leq x_j \leq 1; \quad j = 1, 2, \dots, t \end{aligned}$$

When the objection function  $f_1(x) = x_1$  is minimized as described, the solution vector  $(0, 0, \dots, 0, 1)$  of length  $t$  may result. That same solution vector may be found when each  $f_j(x) = x_j$ ,  $j = 1, 2, \dots, t-1$  is minimized. A different solution vector results when  $f_t(x) = x_t$  is minimized, so the entire minimization effort has resulted in just two distinct seeds. We mitigate that effect as follows. First, we scale the objective functions so that their values over the feasible region are of the same magnitude. Second, for each  $j$  and suitably large value  $M$ , we minimize

$f_j(x) + M \sum_{i \neq j} f_i(x)$ . The objective vector so determined for  $j$  tends to be a Pareto vector where  $f_j(x)$  has maximum value and thus may well be different from the other solutions. This is so for the above example MOP.

If a few added seeds help much, why not add some more? It is well known [22] that for hard MOPs this may not be possible if weighted sums of the objective functions are minimized to obtain seeds, as done here. We include a simple proof for completeness. Let  $P$  be the Pareto front and denote by  $C$  the nonnegative orthant of the space of objective vectors. Let  $P + C$  be the set obtained by pairwise addition of the vectors of  $P$  and  $C$ . Thus,  $P + C = \{u \mid u = p + c; p \in P, c \in C\}$ . Minimization of positive-weight sums of the objective functions can only produce objective vectors of  $P$  that lie on the boundary of the convex hull  $D$  of  $P + N$ . For a hard MOP, it is entirely possible that the seeds computed by the above process are the only points of  $P$  that are on the boundary of  $D$ . The above sample MOP is an instance. Given these considerations, we refrain from time-consuming and possibly futile computations looking for additional seeds. This argument applies particularly to cold-start cases with black-box evaluations where we have no clue about problem structure.

In our implementation, we use the optimal solutions of the single-function minimization solutions as seeds as described. But we also collect during that phase all objective vectors  $x$  evaluated during the single-function minimization phase, then label each vector as "accept" or "reject" depending on how far its objective values are above those associated with the optimal solutions. That is, if for at least one objective the value is very far above the max value occurring in the seeds, then the label is "reject." Otherwise it is an "accept" case. When the initial population is randomly selected, we compute for each randomly selected vector the closest distance to the "accept" and "reject" cases. If the vector is closest to an "accept" vector, then we accept the vector. Otherwise we discard it. Note that the latter decision does not involve any computation of function values.

For the single-function minimization instances, a large variety of methods are available; see [17, 23]. For the computations reported in Section 5, we have used the mesh-search method NOMAD [1, 19] and the direction-finding methods DENCON and DENPAR of [20], which are based on [9].

For the selection of the EA, many candidates exist as well; see the summary of these methods in [10]. As mentioned above, we have used NSGA-II for the computational results of Section 5.

The joint use of single-objective and multiobjective schemes requires some care. The next section describes appropriate modifications of the basic idea.

## 4 Modifications

The simple sequence of computations described above needs to be adapted when hard MOP instances are to be reliably solved in reasonable time. The modifications concern the single-function minimization step; choice of population size in the EA; construction of the initial population; test of convergence and, as part of

that test, recognition of the end of the crystallization phase; and parallelization of computing effort during all steps.

#### 4.1 Finding a Feasible Solution

It is quite possible that the selected method for minimizing single objective functions of MOP may fail to find a feasible solution. In that case, we do not try other single-function minimization schemes in the hope that one of them succeeds, since this may lead to a long and fruitless effort. Instead, it seems prudent to invoke a scheme that just looks for a feasible solution and has proved to be reliable. In computational tests involving difficult MOP instances arising in Engineering, we have found that the NSGA-II used here as EA for multiobjective minimization is also effective for finding feasible solutions in those hard cases. So we revert to the latter algorithm. If the problem turns out to be infeasible, the algorithm has a tendency to point to the constraint(s) that cause the infeasibility, a useful feature. If a feasible solution is found, the selected single-function minimization method starts with that solution.

#### 4.2 Multi-stage Single-Function Minimization

To assure that the seeds are extreme points of the Pareto Front or are at least close to these points, we want the seeds to be close-to-optimal if not optimal solutions for the single objective function cases.

For hard Engineering problems, we strive for that goal with a two-stage method for the single function minimization cases. In the first step, we execute a method that likely does not get trapped by locally but not globally optimal solutions. It suffices that the method finds a point near a globally optimal solution. Then we apply a second method that likely finds a globally optimal solution when the starting point is near one such solution. Here, we use NOMAD [1, 19] as first method and DENPAR [20] as second one.

Now suppose the single-function minimization problems are known or anticipated to be easy. Correspondingly, a comparatively simple and fast method should be used. Here we employ DENCON [20], which uses coordinate search directions plus other directions that asymptotically search out the entire space. In particularly easy cases, we restrict DENCON to coordinate directions. Below, we call that version DENCON-R, where the R stands for "restricted."

#### 4.3 Population Size for EA

The size of the population for the EA is decided according to several considerations.

Generally speaking, larger populations tend to produce a more detailed representation of the Pareto front, but may also require a larger evaluation effort by the black box. Of course, parallelization can mitigate that negative effect.

So let's consider the decision in the absence of parallelization. Then we would want a population large enough so that (1) the EA converges to points of the

Pareto front, and (2) the representation of the Pareto front is sufficiently fine-grained.

Experiments with NSGA-II have shown that convergence often can be achieved with a comparatively small population, but may produce an insufficiently detailed representation of the Pareto front. Put differently, convergence dictates population size to a lesser extent than precision of the Pareto front representation does.

Suppose we use a population size that is just large enough for convergence. To obtain a fine-grained representation nevertheless, we store and later retrieve all solution vectors produced by the EA. Since we demand a certain accuracy in the final solution as part of the convergence conditions, we obtain a reasonably fine-grained representation of the Pareto front.

There is a second way to improve the Pareto front representation. Specifically, we enforce bounds on the objective functions as the crystallization process creates portions of the Pareto front starting at the seeds. We call the process *dynamic limit*. Effectively, we shift the exploration of space by the EA to regions where as yet we do not have points of the Pareto front. As the shifting occurs, the tightening bounds eliminate Pareto points outside the bounds. But these points have already been stored. When the process stops, we use all stored points plus those of the final population to construct a representation of the Pareto front.

#### 4.4 Initial Population

Suppose the initial population of EA has size  $N$ , and we want to insert  $t$  seeds while maintaining population size. Thus,  $t$  points must be removed from the population.

We view this as a clustering problem where  $N$  centers must be selected from  $N + t$  points, and where the  $t$  seeds must be among the centers. Thus, according to the cluster metric, the selected  $N$  points are a best representation of the  $N + t$  points. We use the initialization phase of the `kmeans++` clustering algorithm [2] for this task, since that phase, by itself, has a guarantee of good performance, yet is very fast.

We still need to decide  $N$ . Suppose we have determined some  $m$  to be a lower bound for population size that guarantees convergence for a large number of MOP instances. We then define  $N$  to be the smallest integer greater than or equal to  $m + t$  that for the EA is permissible as population size. In the case of NSGA-II, extensive computational tests have determined that  $m = 40$  can serve as lower bound. Since population size for NSGA-II must be a multiple of 4, we define  $N$  to be the smallest integer  $k \geq 40 + t$  that is divisible by 4. Thus, for the frequently encountered case  $t \leq 4$ , we have  $N = 44$ .

Inserting other, possibly user-supplied points as seeds is handled in an analogous manner. We omit details.

We should mention that [13] recommends the slightly larger value  $m = 52$  for NSGA-II. Evidently, that bound is based on the consideration that NSGA-II should not only converge but also create, just by the final population, a reasonable representation of the Pareto front. Accordingly, we conjecture that the

difference to  $N = 44$  used here for  $t \leq 4$  is due to the fact that the population must just be large enough to assure convergence, see Section 4.3.

#### 4.5 Convergence

Deciding convergence of an EA such as NSGA-II isn't easy; see, for example, the discussion in [11, 24]. We encountered the same difficulties when we applied NSGA-II directly, without computation of seeds. But the computation of seeds supplies information that allows us to assess convergence in a reasonable manner and also identify completion of the crystallization phase. Here are the details.

During the run of NSGA-II we check if the distribution of function vectors near the seeds no longer changes and has become reasonably dense. Once that situation is at hand, we declare that the crystallization phase has been completed.

From then on, we monitor the extent to which new populations dominate old ones. When that extent shrinks to a selected accuracy threshold, the method stops and declares convergence. During the solution of difficult problems, it may happen that two successive populations differ little, yet convergence has not been achieved. Hence, is important that we do not compare successive populations, and some iterations should elapse before the convergence criteria are applied. Estimation of that elapsed interval is difficult. Generally, the harder the problem, the larger the elapsed interval should be. We estimate the hardness by the total evaluation count required during the single-function minimization phase, and define the size of the elapsed interval via that count.

#### 4.6 Refinement of Incomplete Solutions

It may happen that the representation found for the Pareto front is incomplete, indicated by gaps or unexpected jumps of the function values. A *cutout option* allows us to investigate such a region, as follows.

First, we add suitable lower bounds on objective values to focus the search on the region of interest.

Second, we use the lower bounds to reduce the collection of solution vectors on hand to eliminate all solutions whose objective values violate at least one of the bounds. Of the remaining vectors, we select for each objective function one vector that minimizes the objective and one vector that maximizes the objective.

Third, we apply NSGA-II as before, except that solutions whose objective values violate bounds are considered to be infeasible.

#### 4.7 Parallelization

If the evaluation of objectives and constraint functions requires little computational effort, then there is no need for parallelization. But if that task is done by an expensive black box, then parallelization is essential. Indeed, for black box times in the range of several seconds per evaluation, the entire method must

terminate after at most a few thousand evaluation cycles if it is to be of practical use. In the more severe case of evaluation times of several minutes, at most a few hundred evaluations are acceptable.

Parallelization of computations for an EA is trivial if  $p$  is below the bound on population size decided for the EA as described in Section 4.4, which for the example cases described in this paper is 44 for NSGA-II. Hence we consider here  $p = 44$ .

Of course, the single-function minimization methods must also be able to use  $p$  processors in parallel. Both NOMAD [1, 19] and DENPAR [20] selected here reasonably satisfy that requirement. Specifically, for an MOP with  $n$  variables, NOMAD can utilize up to  $2n$  processors, while DENPAR can make use of up to  $32n$  processors.

## 5 Computational Results

We demonstrate various aspects and results mentioned in the above discussion, beginning with the crystallization claims of Section 3.

### 5.1 Example Problem zdt1

We use the well known problem zdt1 [6] to demonstrate the case where the entire solution process consists just of the crystallization phase. The problem has 30 variables and 2 objective functions.

The single-function minimization cases of zdt1 are so easy that DENCON-R suffices for solution. The method requires a total of 800 evaluations to produce two seeds. Here and later evaluation counts are always rounded to the nearest 100.

The initial population for NSGA-II is obtained by inserting the two seeds into a randomly selected population of size 44, as discussed in Section 4.3.

We then run NSGA-II. Here and in all subsequent runs the following values are used for the NSGA-II parameters: crossover probability  $p_c = 0.9$ , mutation probability  $p_m = 1/n$  where  $n$  is the number of variables; crossover distribution index  $\eta_c = 20$ , and mutation distribution index  $\eta_m = 20$ . These values are identical to those proposed in [6].

We stop NSGA-II after 700 evaluations and collect all undominated solutions. For comparison, we also run NSGA-II just starting with a randomly selected population of size 44, and again stop after 700 evaluations. Finally, we run the original NSGA-II for 40,000 evaluations, with population size 200, to get an estimate of the Pareto front. Fig. 1 displays the results. A red triangle corresponds to a solution point when seeds are used, while a black square refers to the case without seeds. The optimal solution is shown by small black dots. NSGA-II when started with two seeds has managed to grow a portion of the estimated Pareto front from the seeds. In contrast, the standard version initialized with just a randomly selected population has not yet detected any part of the estimated Pareto front.

We continue the run and halt the process again after a total of 1,500 evaluations; see Fig. 2. NSGA-II when supplied with two seeds has managed to create a large portion of the estimated Pareto front. In contrast, the run of NSGA-II without seeds still has not created any part of the estimated Pareto front.

We let the seed case run till convergence is determined as discussed in Section 4.5. This happens after 2,400 evaluations. At that time, a fine-grained representation of the Pareto front has been produced. The unassisted NSGA-II run still does not have any values of the estimated Pareto front; see Fig 3.

The reader may argue that the comparison of NSGA-II with and without seeds should allow the case without seeds to make use of evaluations needed for the computation of the seeds. We have ignored that aspect so far to exhibit the effect of the seeds. Now let's add in the 800 evaluations needed for computation of the two seeds, so the two cases with and without seeds carry out the same total number of evaluations. When NSGA-II is run without seeds for a total of  $2,400 + 800 = 3,200$  instead of 2,400 evaluations and the resulting data are inserted into Fig. 3, we get Fig 4, where the values for the case without seeds are still far from the Pareto front.

The next two sections describe the solution process for harder problems where the crystallization phase creates a surface that isn't yet the Pareto front.

## 5.2 Problem with Sphere Functions

We go to a harder MOP instance with two functions based on the sphere function  $\sum_j x_j^2$ . Just minimizing one sphere function is trivial for most if not all single minimization algorithms, in particular for NOMAD, DENPAR, DENCON, and DENCON-R. But this isn't the case for NSGA-II; see [7]. Here is the MOP, called where  $d$  and  $n$  are positive integers.

$$\min \left\{ \sum_{j=1}^n (x_j + d)^2, \sum_{j=1}^n (x_j - d)^2 \right\}$$

$$-50 \leq x_j \leq 100; j = 1, 2, \dots, n$$

The first objective function attains the minimum of 0 when, for all  $j$ ,  $x_j = -d$ . The second objective function does so when, for all  $j$ ,  $x_j = d$ . For the demonstration, we use the case of  $n = 20$  and  $d = 10$ . We call the resulting problem *quad2*.

Single function minimization by DENCON-R produces optimal solutions for the two objective functions using a total of 2,000 evaluations.

With the resulting two seeds, we start NSGA-II and halt computation after just 400 evaluations. At that time, the solutions cover the full range of function values, so we have reached the end of the crystallization phase. But the convergence criteria are far from being met. For comparison, NSGA-II run without seeds for 400 evaluations produces just six undominated solutions that are far

above the solutions obtained with seeds. Finally, we run the original NSGA-II for a large number of evaluations, with population size 100, to obtain an estimate of the Pareto front. Indeed, it takes 400,000 evaluations to get the desired result since the method has difficulty computing the Pareto front near the function extremes. Fig. 5 depicts the results of the three runs.

When we continue the run with seeds, we have convergence after 12,100 evaluations. We also continue the run of NSGA-II without seeds and stop the process also after a total of 12,100 evaluations. Fig. 6 displays the results. The case without seeds does not represent significant parts of the estimated Pareto front near the extreme points. When seeds are used, the entire estimated Pareto front is covered with high precision.

For a final comparison allowing the same total number of evaluations for the case with and without seeds, we run NSGA-II without seeds once more, increasing the total of evaluations from 12,100 to  $2,000 + 12,100 = 14,100$ . Fig. 7 has the results. Evidently, the case without seeds still does not present parts of the estimated Pareto front near the extreme points.

### 5.3 Difficult Problem *maxzkv*

We demonstrate the effect of seeds for a case where minimization of the single objective functions isn't easy even for methods specialized for that task. For this purpose, we combine the well-known and reasonably difficult problems *maxl* [8] and *zkv20* [18], both of which have 20 variables.

We assign -50 and 100 as lower and upper bound to each variable, and scale the variables of *zkv20* by a factor 10. Finally, we shift each entry of the optimal vector of *maxl* by a constant 10 so that the optimal solutions of the revised *maxl* and the revised *zkv20* do not coincide. We call the resulting problem *maxzkv*.

The single-function minimization cases producing the seeds are so difficult that application of DENCON-R does not suffice for solution. Indeed, we use the hybrid method where NOMAD is followed by DENPAR. The hybrid method requires 12,300 evaluations and determines four seeds instead of just two, since each of NOMAD and DENPAR produces two nondominated solutions. We obtain an estimate of the Pareto front by running NSGA-II with the four seeds for 500,000 evaluations, using population size 100 for a fine-grained representation. We cannot avoid the use of seeds since NSGA-II without seeds for the same number of evaluations fails to determine a large portion of the estimated Pareto front. Fig. 8 shows the estimated Pareto front and also the output of the failed run without seeds.

We again run NSGA-II with seeds, but stop when we reach the end of the crystallization phase after 3200 evaluations. For comparison, we also run NSGA-II starting without seeds, and again stop after 3200 evaluations. Fig. 9 displays the results. As before, a red triangle corresponds to a solution point when seeds are used, while a black square refers to the case without seeds. The estimated optimal solution is shown by black dots. While the solution produced with seeds covers the entire range of the estimated Pareto front, with a few points near the

extremes lying on that front, the solution obtained without seeds just has seven points, all lying outside the estimated Pareto front.

We let the seed case run till the convergence condition discussed in Section 4.5 is satisfied. This happens after 42,000 evaluations. At that time, a reasonable representation of the estimated Pareto front has been produced; see Fig 10. The unassisted NSGA-II run still does not have any values for a large portion of the estimated Pareto front. The latter result is not unexpected, given the earlier discussion of Fig 8.

For a final comparison allowing the same total number of evaluations for the case with and without seeds, we run NSGA-II without seeds once more, increasing the total of evaluations from 42,000 to  $12,300 + 42,000 = 54,300$ . Fig. 11 has the results. The conclusion about NSGA-II running without seeds is essentially unchanged.

Recall from Section 3 that during the computation of seeds we collect all evaluated vectors and classify them as "accept" or "reject" depending on how far their objective values are above the objective values of the seeds. Further, during selection of the initial population, a randomly selected vector is accepted for the population if its distance to the "accept" vectors is less than the distance to the "reject" vectors. The latter decision does not involve any computation of function values. When the initial population for maxzkv is selected, a total of 1,234 vectors are randomly generated, of which 1,190 are rejected; as expected, 44 are accepted.

To show the contribution of this selection process to convergence, in particular, to the crystallization phase, we compare two runs. The first one has been described above, where the crystallization phase is reached after 3,200 evaluations. Using that same limit on the number of evaluations, we run the case again, but this time suppress the collection of "accept" and "reject" data. Instead, we just randomly select 44 vectors for that population. Of course, we still insert the seeds into that population as usual. Fig. 12 compares the solutions obtained that way. The case without "accept" and "reject" data is somewhat inferior to the case where these data are collected and used.

We should emphasize that the criteria for selecting "accept" and "reject" data as well as other decision parameters were developed by evaluating a number of problems that did not include the example cases explicitly cited in the paper.

Finally, we demonstrate the effectiveness of parallelization for maxzkv, using  $p = 44$  processors. The computation of seeds by NOMAD and DENPAR requires a total of 900 evaluation cycles of the 44 processors. The subsequent construction of the Pareto front uses another 1500 cycles until convergence is claimed. Thus, the entire solution requires a total of 2,400 cycles. In comparison, the computations done earlier with single processors used 54,300 evaluations, and thus 54,300 single-processor cycles. Since we used 44 processors, we thus have an overall utilization of  $(54300/2400)/44 = 51\%$ . Of course, NSGA-II runs with perfect efficiency. But even the utilization by NOMAD and DENPAR, which is  $(12300/900)/44 = 31\%$ , is reasonable.

We emphasize that solution trajectories produced by NOMAD and DENPAR typically change when the number of processors changes. But the seeds are very similar and propel NSGA-II to produce a virtually identical approximation of the Pareto front; Fig 13 demonstrates this fact.

## References

1. M. Abramson, C. Audet, G. Couture, J. Dennis, Jr., S. Le Digabel, and C. Tribes. The NOMAD project. Software available at <http://www.gerad.ca/nomad>.
2. D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '07)*, pages 1027 – 1035, 2007.
3. J. Branke, K. Deb, K. Miettinen, and R. Slowinski. *Multiobjective Optimization*. Springer-Verlag, 2008.
4. K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, part II: Handling constraints and extending to an adaptive approach. Technical Report 2012010, Indian Institute of Technology Kanpur, 2012.
5. K. Deb and H. Jain. An improved NSGA-II procedure for many-objective optimization, part I: Problems with box constraints. Technical Report 2012009, Indian Institute of Technology Kanpur, 2012.
6. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, 6:182 – 197, 2002.
7. K. Deb, K. Sindhya, and T. Okabe. Self-adaptive simulated binary crossover for real-parameter optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2007)*, UCL London, pages 1187 – 1194. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2007), UCL London, 2007.
8. G. Di Pillo, L. Grippo, and S. Lucidi. A smooth method for the finite minimax problem. *J. Glob. Optim.*, 60:187 – 214, 1993.
9. G. Fasano, G. Liuzzi, S. Lucidi, and F. Rinaldi. A linesearch-based derivative-free approach for nonsmooth optimization. *IASI-CNR Working Paper*, 2013.
10. M. Gen and L. Lin. Multiobjective evolutionary algorithm for manufacturing scheduling problems: state-of-the-art survey. *J. Intell. Manuf.*, DOI 10.1007/s10845-013-0804-4, 2013.
11. T. Goel and K. Deb. Hybrid methods for multi-objective evolutionary algorithms. In *Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning (SEAL,02)*. (Singapore), pages 188 – 192, 2002.
12. C. Grosan and A. Abraham. Hybrid evolutionary algorithms: methodologies, architectures, and reviews. In *Hybrid evolutionary algorithms*, pages 1 – 17. Springer, 2007.
13. A. Hernandez-Diaz, C. A. Coello Coello, F. Perez, R. Caballero, J. Molina, and L. Santana-Quintero. Seeding the initial population of a multi-objective evolutionary algorithm using gradient-based information. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence)*, pages 1617–1624, 2008.
14. B. Huang, B. Buckley, and T.-M. Kechadi. Multi-objective feature selection by using NSGA-II for customer churn prediction in telecommunications. *Exp. Sys. Appl.*, 37:3638 – 3646, 2010.

15. H. Jain and K. Deb. An improved adaptive approach for elitist nondominated sorting genetic algorithm for many-objective optimization. In *Evolutionary Multi-Criterion Optimization*, pages 307–321, 2013.
16. S. Kannan, S. Baskar, J. D. McCalley, and P. Murugan. Application of nsga-ii algorithm to generation expansion planning. *IEEE Trans. Power Sys.*, 24:454 – 462, 2009.
17. N. Karmitsa, A. Bagirovb, and M. M. Mäkelä. Comparing different nonsmooth minimization methods and software. *Opt. Meth. Softw.*, 27:131 – 153, 2013.
18. M. Laguna, J. Molina, F. Pérez, R. Caballero, and A. G. Hernández-Díaz. The challenge of optimizing expensive black boxes: a scatter search/rough set theory approach. *J. Oper. Res. Soc.*, 61:53 – 67, 2009.
19. S. Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Trans. Math. Softw.*, 37:1 – 15, 2011.
20. G. Liuzzi and K. Truemper. Two parallelized versions for linesearch-based derivative-free optimization algorithms. Technical report, IASI-CNR, Italy, 2014.
21. M. Lozano and C. García-Martínez. Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report. *Comp. & Op. Res.*, 37:481 – 497, 2010.
22. R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26:369–395, 2004.
23. L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *J Glob. Optim.*, 56:1247 – 1293, 2013.
24. K. Sindhya, K. Deb, and K. Miettinen. Improving convergence of evolutionary multi-objective optimization with local search: a concurrent-hybrid algorithm. *J Nat. Comput.*, 10:1407 – 1430, 2011.
25. A. Sinha and D. E. Goldberg. A survey of hybrid genetic and evolutionary algorithms. *IlligAL report*, 2003004, 2003.
26. H. Soyel, U. Tekguc, and H. Demirel. Application of nsga-ii to feature selection for facial expression recognition. *Comput. and Elec. Eng.*, 37(6):1232 – 1240, 2011.
27. E. Zitzler, E. Laumanns, and L. Thiel. Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Proceedings of the EU-ROGEN 2001-Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems*, pages 95 – 100, 2001.

Figures

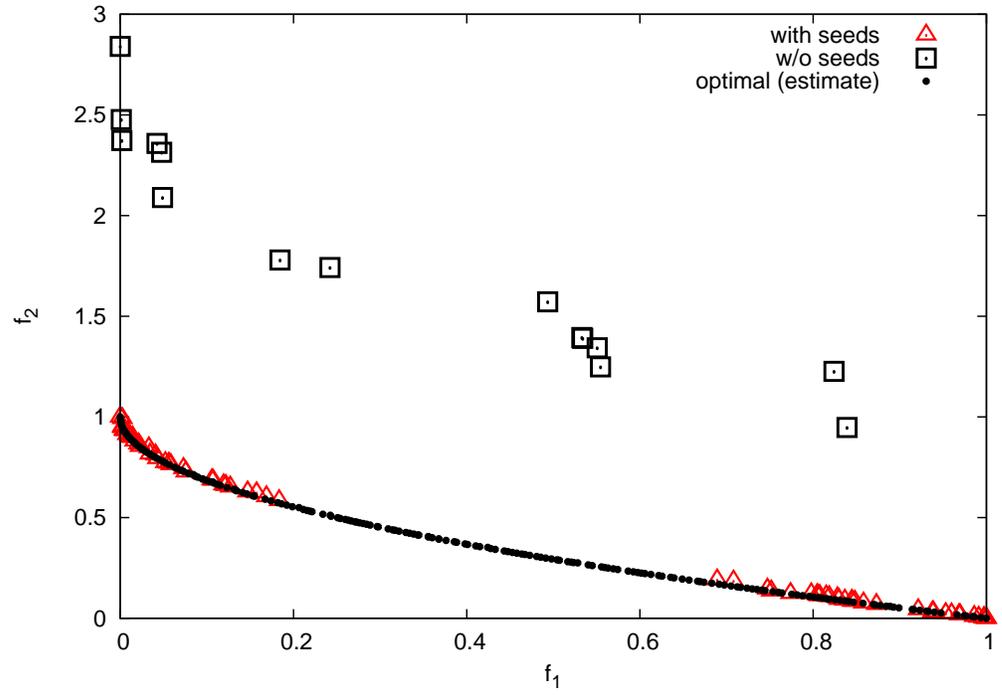


Fig. 1. zdt1: 700 evaluations with and without seeds

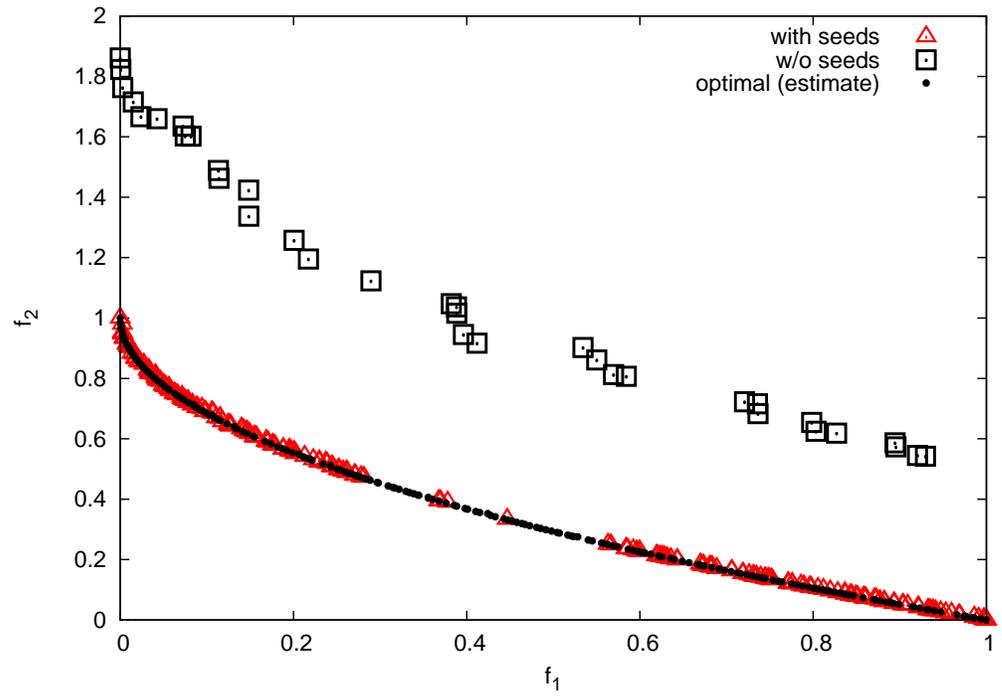


Fig. 2. zdt1: 1,500 evaluations with and without seeds

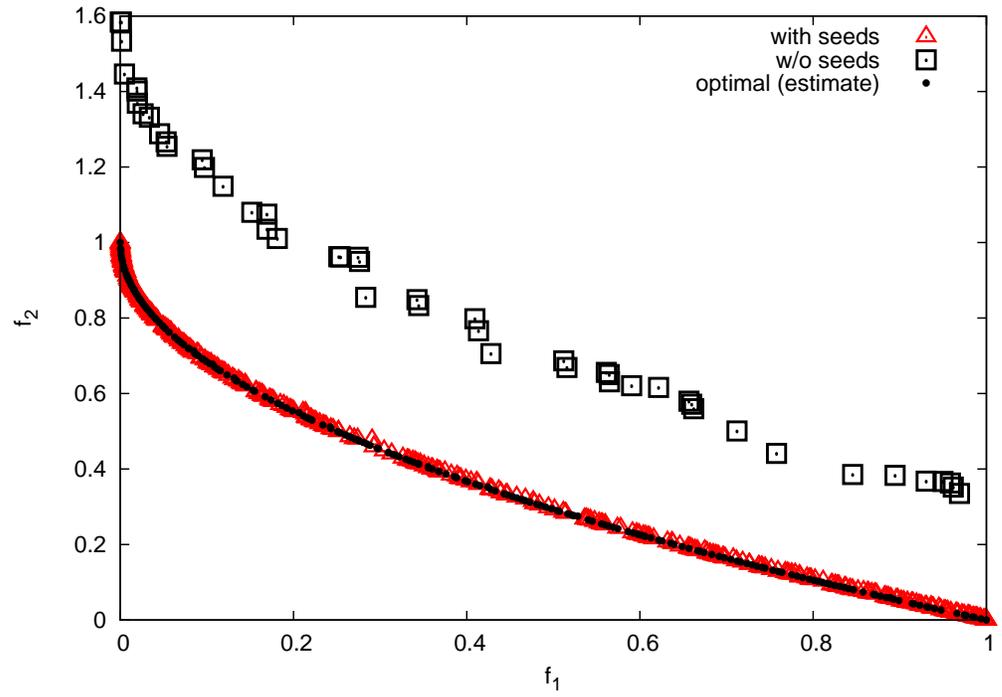


Fig. 3. zdt1: 2,400 evaluations with and without seeds

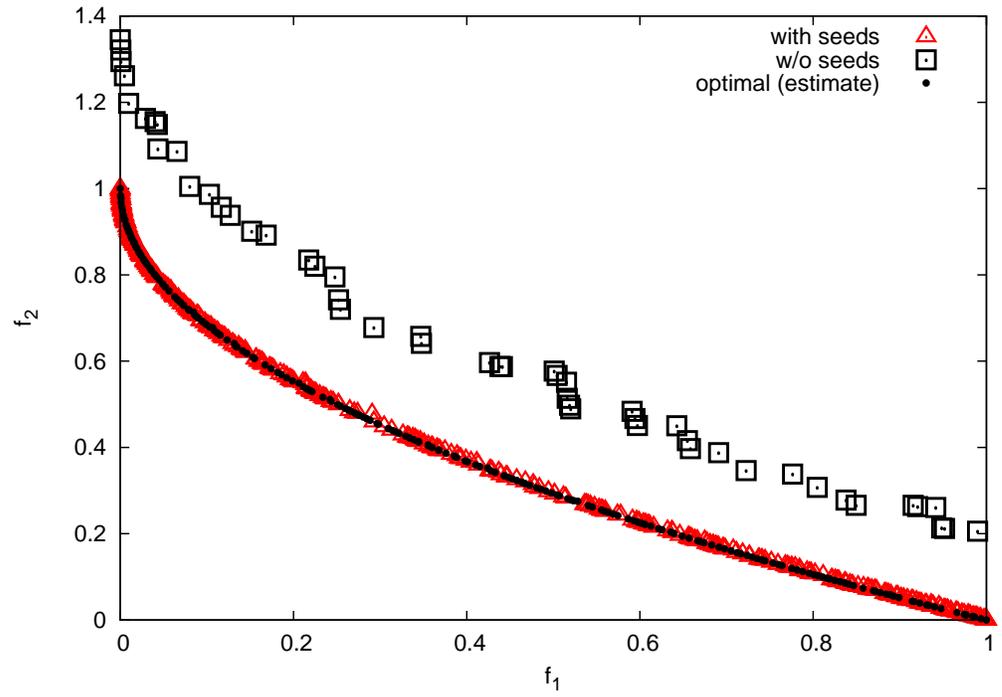


Fig. 4. zdt1: 3,200 evaluations with and without seeds

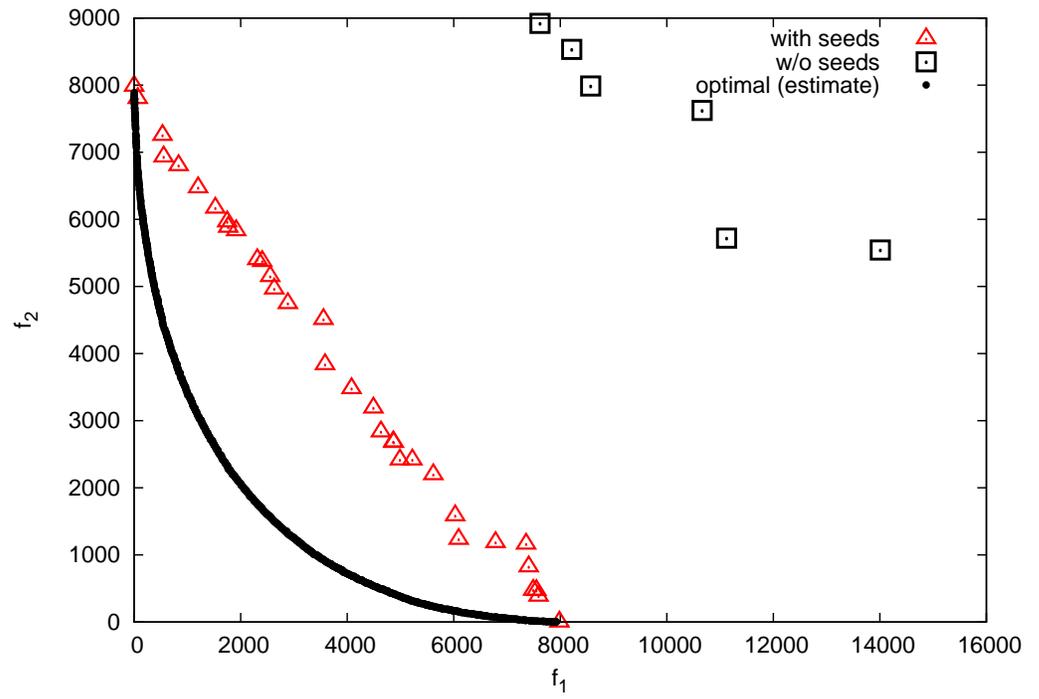


Fig. 5. quad2: 400 evaluations with and without seeds

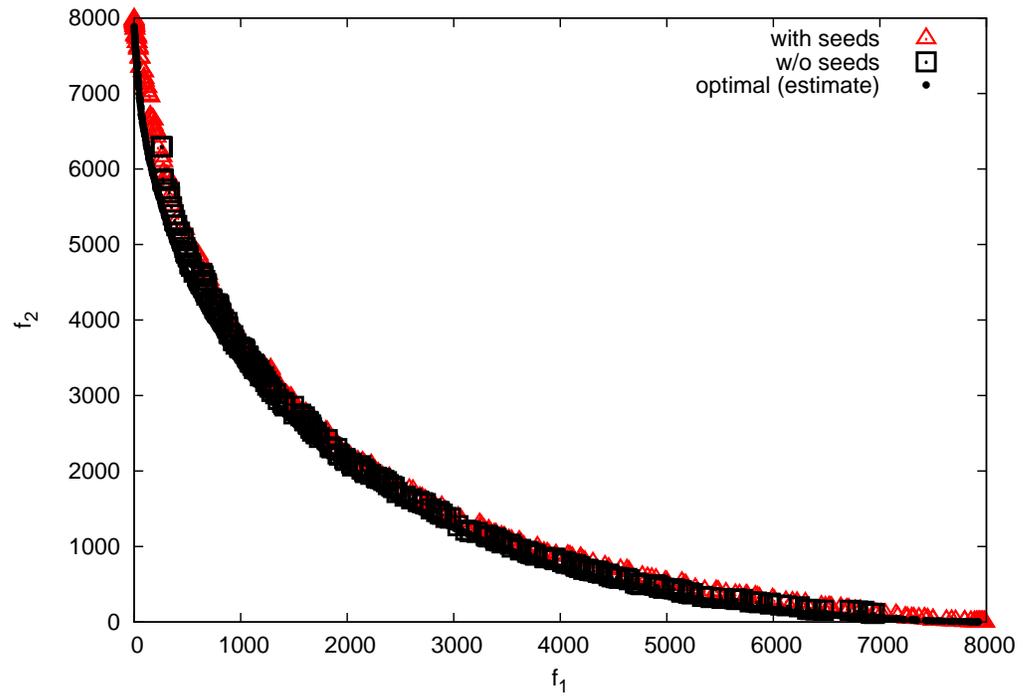


Fig. 6. quad2: 12,100 evaluations with and without seeds

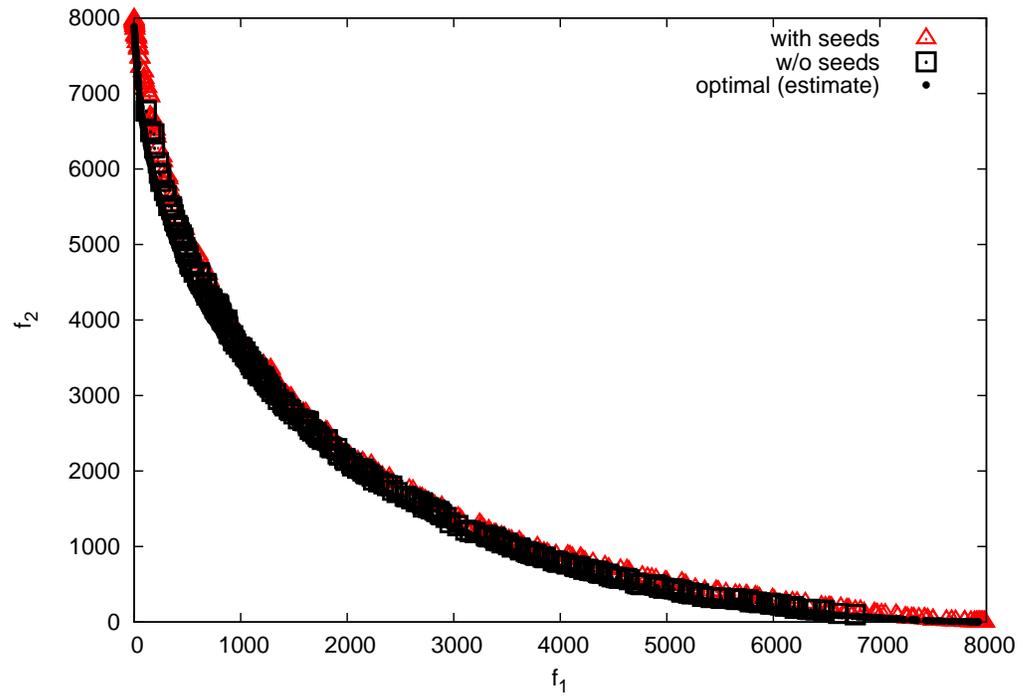


Fig. 7. quad2: 14,100 evaluations with and without seeds

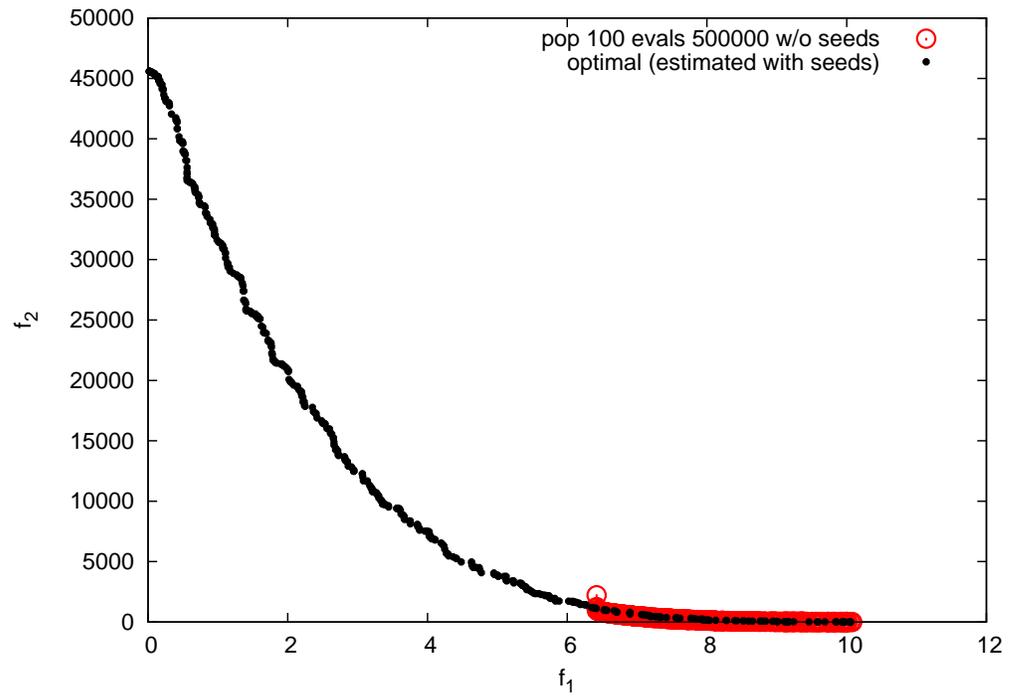


Fig. 8. maxzkv: Pareto front estimated with and without seeds

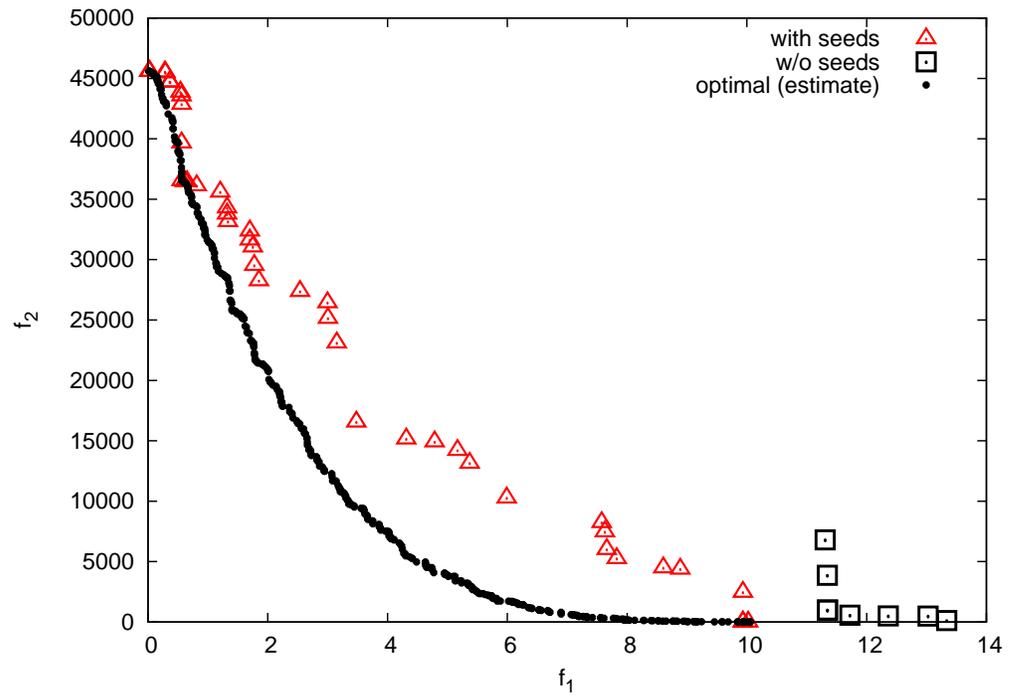


Fig. 9. maxzkv: 3,200 evaluations with and without seeds

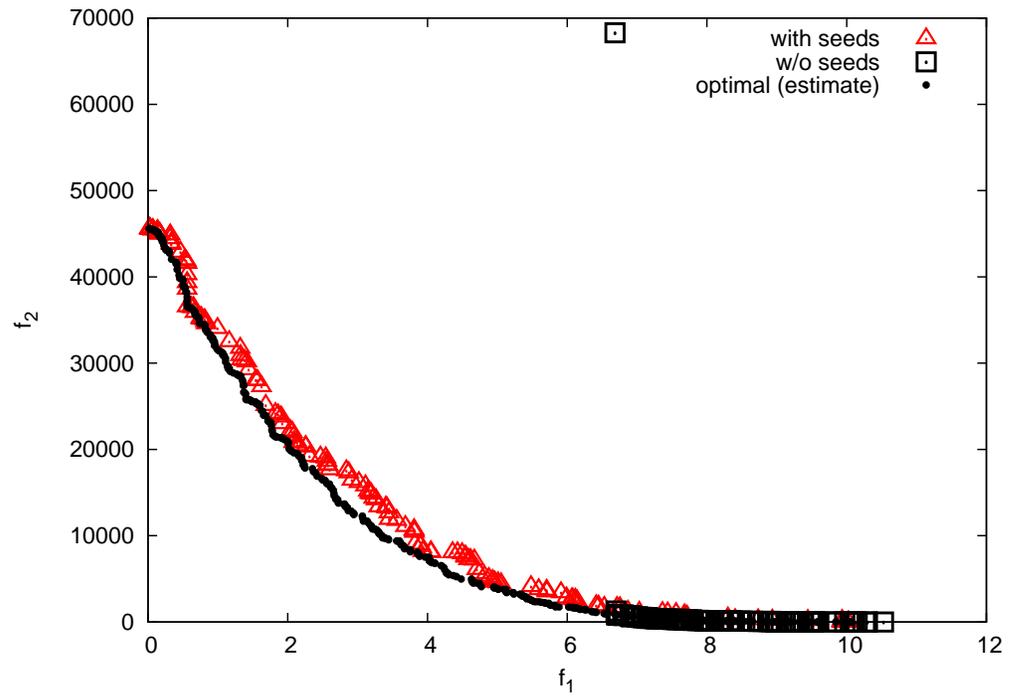


Fig. 10. maxzkv: 42,000 evaluations with and without seeds

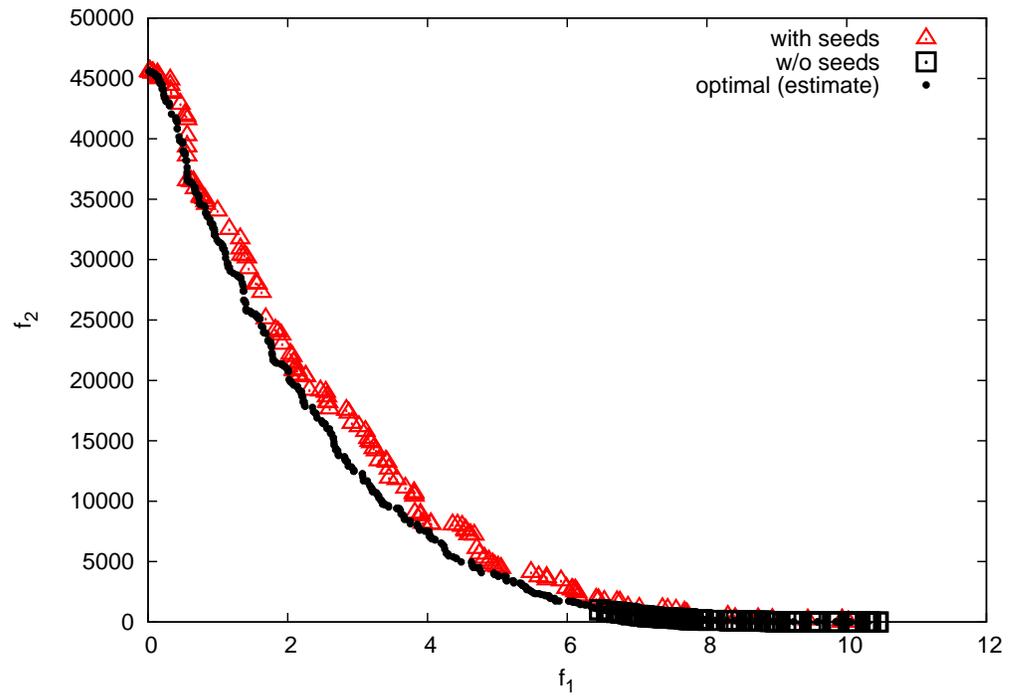


Fig. 11. maxzkv: 54,300 evaluations with and without seeds

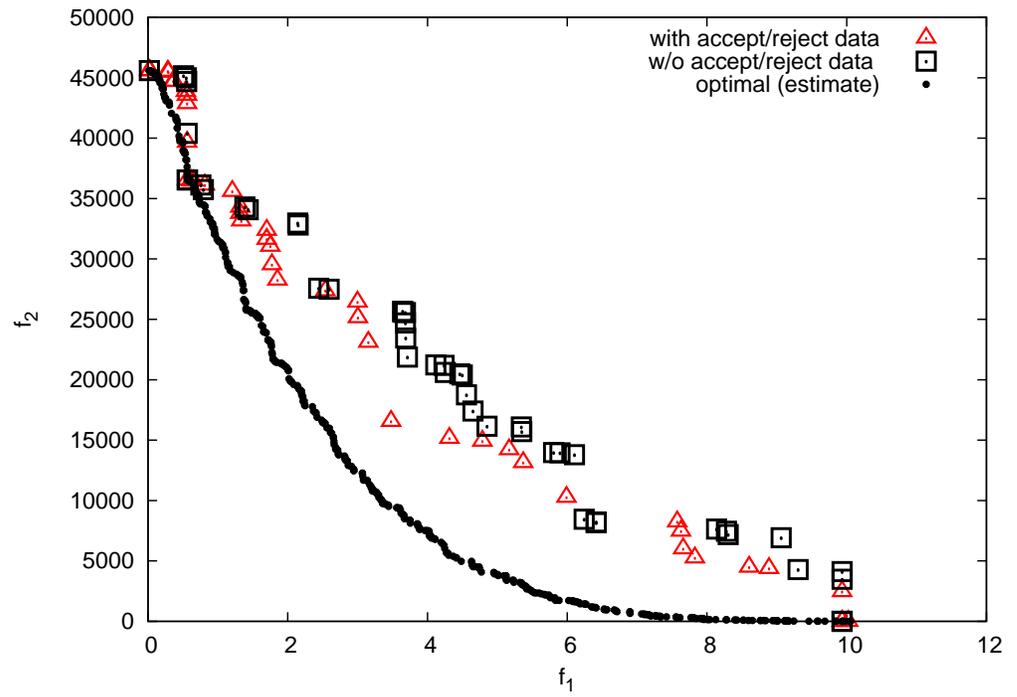


Fig. 12. maxzkv: 3,200 evaluations with and without use of accept/reject data

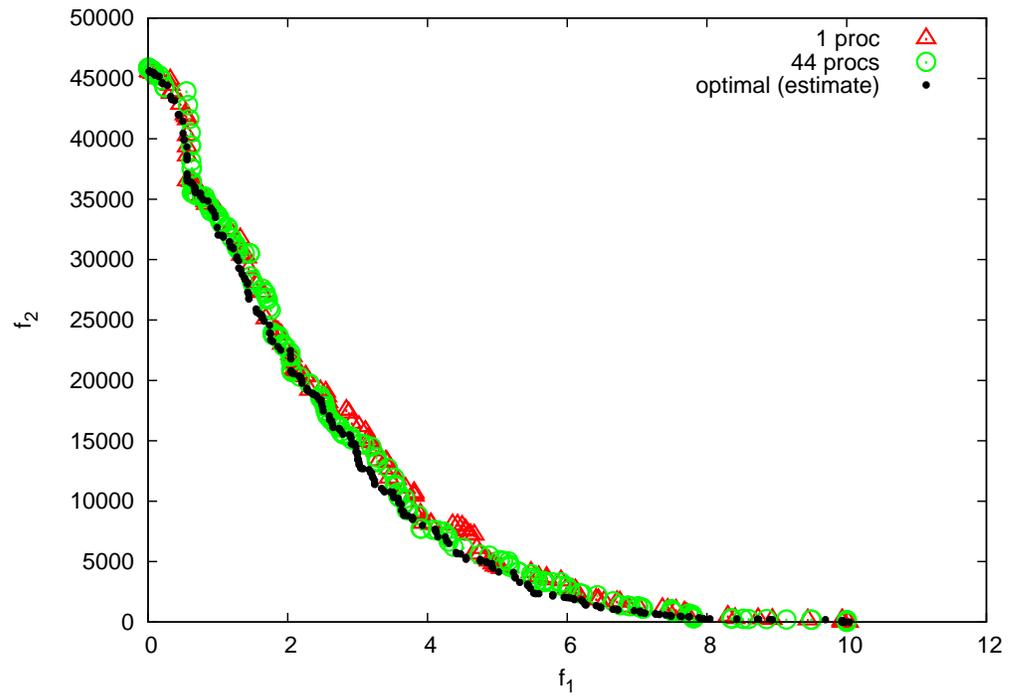


Fig. 13. maxzkv: Solution obtained with 1 versus 44 processors