Omni-optimizer: A Procedure for Single and Multi-objective Optimization

Kalyanmoy Deb and Santosh Tiwari

Kanpur Genetic Algorithms Laboratory (KanGAL),
Indian Institute of Technology Kanpur,
Kanpur, PIN 208 016, India
{deb, tiwaris}@iitk.ac.in
http://www.iitk.ac.in/kangal

Abstract. Due to the vagaries of optimization problems encountered in practice, users resort to different algorithms for solving different optimization problems. In this paper, we suggest an optimization procedure which specializes in solving multi-objective, multi-global problems. The algorithm is carefully designed so as to degenerate to efficient algorithms for solving other simpler optimization problems, such as single-objective uni-global problems, single-objective multi-global problems and multi-objective uni-global problems. The efficacy of the proposed algorithm in solving various problems is demonstrated on a number of test problems. Because of it's efficiency in handling different types of problems with equal ease, this algorithm should find increasing use in real-world optimization problems.

1 Introduction

With the advent of new and computationally efficient optimization algorithms, researchers and practitioners have been attempting to solve different kinds of search and optimization problems encountered in practice. One of the difficulties in solving real-world optimization problems is that they appear in different forms and types. Some optimization problems may have to be solved for only one objective, some other problems may have more than one conflicting objectives, some problems may be highly constrained, and some may have more than one optimal solutions. When faced with such problems, a user first analyzes the underlying problem and chooses a suitable algorithm for solving it. This is because an algorithm efficient for finding the sole optimum in a single-objective optimization problem cannot be adequately applied to find multiple optimal solutions present in another optimization problem. To solve different kinds of problems, a user needs to know different algorithms, each specialized in solving a particular class of optimization problem.

In this paper, we propose and evaluate a single optimization algorithm for solving different kinds of function optimization problems often encountered in practice. The proposed *omni-optimization* algorithm adapts itself to solve different kinds of problems – single or multi-objective problems and uni or multi-global problems. The motivation for developing such a generic procedure came from the generic programming practices. For example, if a programming task is to develop a code for adding a few integers, a generic approach would be to use the following strategy.

```
Add a few integers:
begin
   print ‘enter number of integers to be added’, read $n$
   sum = 0
   for $i = 1$ to $n$
      print ‘enter integer $i$’, read a[$i$]
      sum = sum + a[$i$]
   print ‘Sum =’ sum
end
```

Interestingly, the same code can be used for adding any number of integers initially defined by the variable n. If $n = 1$ is used (thereby trying to add only one number to zero), the code degenerates to printing the same integer as the outcome of the addition. On a similar vein, a generic optimization procedure should find optimal solutions for a multi-objective optimization problem and the same procedure should degenerate to solving a single-objective optimization problem if only one objective function is used. Similarly, our proposed approach can find multiple optimal solutions, if present in a problem, and will automatically degenerate to find the sole optimum of a uni-global optimization problem.

The proposed omni-optimizer is carefully designed to have various properties needed for solving different kinds of optimization problems and is also found to be computationally efficient. The simulation results on 12 test problems show the usefulness of the proposed algorithm and suggest more such studies in the near future.

2 Function Optimization Problems

A function optimization problem may be of different types, depending on the desired goal of the optimization task. The optimization problem may have only one objective function (known as a single-objective optimization problem), or it may have multiple conflicting objective functions (known as a multi-objective optimization problem). Some problems may have only one global optimum, thereby requiring the task of finding the global optimum[1]. Other problems may contain more than one global optima in the search space, thereby requiring the task of finding multiple such global optimal solutions. Although in some optimization

[1] However, in robust optimization tasks, instead of finding the global optimum, the emphasis is on finding a solution which is less sensitive to local perturbation of variables.

tasks, there may be a need of finding the local optimal solutions in addition to finding global optimum solutions, in this study we only concentrate in finding the global optima (one or more) of one or more objective functions.

We consider the following constrained M-objective ($M \geq 1$) minimization problem:

$$\begin{array}{rl} \text{Minimize} & (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x})), \\ \text{Subject to} & g_j(\mathbf{x}) \geq 0, \quad j = 1, 2, \ldots, J, \\ & h_k(\mathbf{x}) = 0, \quad k = 1, 2, \ldots, K, \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, \quad i = 1, 2, \ldots, n. \end{array} \tag{1}$$

A n-variable solution vector $\mathbf{x}$ which satisfies all constraints and variable bounds shown above is called a *feasible* solution. The optimality of a solution depends on a number of KKT optimality conditions which involve finding the gradients of objective and constraint functions [2, 6, 11].

Here, we suggest an omni-optimizer which is capable of finding one or more near-optimal solutions for the following four types of optimization problems in a single simulation run of the algorithm:

1. Single-objective, uni-global optimization problems (the outcome is a single optimum solution),
2. Single-objective, multi-global optimization problems (the outcome is multiple optimal solutions),
3. Multi-objective, uni-global optimization problems (the outcome is multiple efficient points each corresponding to a single Pareto-optimal solution), and
4. Multi-objective, multi-global optimization problems (the outcome is multiple efficient points some of which may correspond to multiple Pareto-optimal solutions).

It is intuitive that the fourth type of optimization problem mentioned above is the most generic one. If designed carefully, an algorithm capable of solving the fourth type of problems can be made to solve other three types of problems in a degenerate sense. The developed algorithm should be capable of solving any of the above problems to its desired optimality without explicitly foretelling the type of problem it is handling. In the following section, we present one such omni-optimizer, which adapts itself to an efficient algorithm automatically for solving any of the above four types of problems.

3 Omni-optimizer

The optimization algorithm starts with an initial population P_0 of size N and an iteration counter t is initialized to zero. In each iteration, a bigger population R_t is constructed by two random orderings of the same population P_t. Thereafter, two binary-tournament selection operations are performed to select two parent solutions. The tournament selection operator prefers feasible solutions over infeasible solutions (constraint handling), non-dominated solutions over dominated solutions (multiple objective handling) and less-crowded solutions over more-crowded solutions (maintenance of diversity). The two parent solutions are then recombined and mutated to obtain two offspring solutions. Any standard genetic operator for each of these operations can be used here. The offspring solutions are included in the offspring population Q_t.

An elite-preservation is performed by combining both parent P_t and offspring Q_t populations together and then by ranking the combined population from best class of solutions to the worst. This way, a good parent solution is allowed to remain in the subsequent population, in case enough good offspring solutions are not created. The ranking procedure uses a modified domination principle (ϵ-domination) to classify the entire combined population into different classes. The best solutions of the population are stored in F_1, the next-best solutions are stored in F_2 and so on.

Now, to create the next population P_{t+1} of size N, we start accepting classes from the top of the list (F_1 onwards) and stop to the class (F_L) which cannot be completely accommodated to P_{t+1} due to size restriction. Then, based on crowding of the solutions of F_L in both objective and decision variable space, we select only those many solutions which will fill the population P_{t+1}. This completes one iteration of the proposed omni-optimizer.

Readers familiar with the elitist non-dominated sorting GA (NSGA-II) [5] may find the proposed omni-optimization procedure similar to that of NSGA-II with some differences. Thus, the proposed procedure is expected to solve multi-objective optimization problems in a manner similar to NSGA-II. The proposed procedure is also capable of solving other kinds of optimization problems mentioned in the previous section. Later, we shall discuss how this procedure degenerates to solve single-objective uni-global and multi-global problems. Here, we first present the omni-optimization procedure as a pseudo-code.

```
The omni-optimization procedure:
begin
   Initialize($P_0$)
   $t = 0$  // iteration counter
   repeat
      $R_t$ = Shuffle($P_t$) $\cup$ Shuffle($P_t$)
      for $i = 1$ to $N-1$ step 2
          // two selection operations
          parent1 = tournament($R_t(2i-1)$, $R_t(2i)$)
          parent2 = tournament($R_t(2i+1)$, $R_t(2i+2)$)
          // crossover and mutation operators
          (offspring1, offspring2) = variation(parent1, parent2)
          $Q_t(i)$   = offspring1
          $Q_t(i+1)$ = offspring2
      $R_t$ = $P_t \cup Q_t$   // elite preservation
      ($F_1$, $F_2$, ...) = ranking($R_t$) // best class $F_1$ and so on
      $P_{t+1}$ = $\emptyset$
      $j = 1$  // class number
      while $|P_{t+1} \cup F_j| \leq N$
         $P_{t+1}$ = $P_{t+1} \cup F_j$  // include classes from best
         crowd_dist(F$_j$)  // crowding distance of each soln.
         $j = j+1$
      $L = j$   // last class to be included partially
      rem = $N$ - |P$_{t+1}$|   // remaining solutions to be filled
      sorting(crowd_dist(F$_L$)) // sort F$_L$ in decreasing order
                                                  of crowding distance
      P$_{t+1}$ = P$_{t+1}$ $\cup$ F$_L$(1:rem) // include top solutions
      $t = t+1$    // increment iteration counter
   until (termination)
end
```

The operator `shuffle(P`$_t$`)` makes a random ordering of the population members of P_t. The `tournament(a,b)` operator compares two solutions `a` and `b` and declares the winner. The following algorithm is used for this purpose:

```
winner = tournament(a,b)
begin
   if a is feasible and b is infeasible, winner = a
   else if a is infeasible and b is feasible, winner = b
   else if both a and b are infeasible,
        winner = (if CV(a) < CV(b)) ? a : b
   else if both a and b are feasible,
        if a dominates b, winner = a
        else if b dominates a, winner = b
        else if crowd_dist(a) > crowd_dist(b), winner = a
        else if crowd_dist(a) < crowd_dist(b), winner = b
        else winner = random_choose(a,b)
end
```

The function `CV(a)` calculates the overall normalized constraint violation of solution `a`, as follows: $\mathtt{CV}(\mathtt{a}) = \sum_{j=1}^{J}\langle\overline{g}_j(\mathtt{a})\rangle + \sum_{k=1}^{K}|\overline{h}_k(\mathtt{a})|$, where $\langle\cdot\rangle$ is a bracket operator [2]. Here, the solution `a` dominates the solution `b` for an M-objective minimization problem, if following conditions are met:

1. $f_i^{\mathtt{a}} \leq f_i^{\mathtt{b}}$ for all $i = 1, 2, \ldots, M$,
2. $f_i^{\mathtt{a}} < f_i^{\mathtt{b}}$ for at least one $i \in \{1, M\}$.

This is the usual *domination* principle used in multi-objective optimization [11, 4]. The `random_choose(a,b)` selects `a` or `b` randomly. The procedure for computing the crowding distance metric `crowd_dist()` is discussed later.

The variation operator takes two solutions and performs genetic operations (such as crossover followed by mutation) and creates two offspring solutions.

The `ranking(R`$_t$`)` ranks the population R_t into different classes (from best to worst) depending on how good the solutions are. The following pseudo-code can be used for this purpose:

```
(F_1, F_2, ...) = ranking(R_t)
begin
   $k = 1$ // class counter
   repeat
      F_k = ∅
      for $i = 1$ to |R_t|
         for $j = 1$ to |R_t| and $j \neq i$
             if R_t($j$) $\epsilon$-dominates R_t($i$), break
         if ($j$ = |R_t|) // R_t($i$) is $\epsilon$-nondominated
             F_k = F_k ∪ R_t(i)
      R_t = R_t \ F_k
      $k = k + 1$
   until (all $2N$ solutions are classified)
end
```

It is not necessary that the repeat-until loop as described above continues till all $2N$ solutions are classified; the loop can be terminated as soon as the last class (L) which can be partially accommodated is encountered.

Solution a ϵ-dominates another solution b if the following conditions are true in an M-objective minimization problem:

1. $f_i^{\mathtt{a}} \leq f_i^{\mathtt{b}}$ for all $i = 1, 2, \ldots, M$,
2. $f_i^{\mathtt{a}} < f_i^{\mathtt{b}} - \epsilon_i$ for at least one $i \in \{1, M\}$.

Here, ϵ_i is a quantity calculated from a user-defined parameter δ, described in equation 2. Figure 1 shows the region which is ϵ-dominated by solution a in a two-objective minimization problem. For simplicity, we use only one user-defined parameter δ as follows:

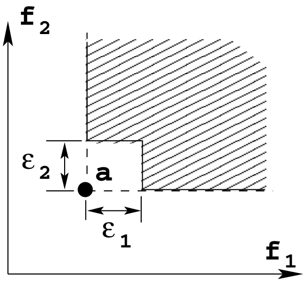

Fig. 1. The ϵ-dominance criterion. Point a ϵ-dominates the shaded region

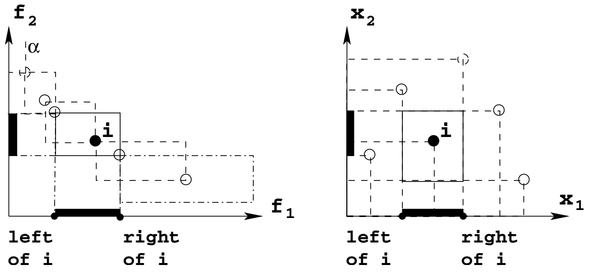

Fig. 2. Objective space and variable space crowding distance computations

$$\epsilon_i = \delta(f_i^{\max} - f_i^{\min}), \tag{2}$$

where $f_i^{\max}$ and $f_i^{\min}$ are the maximum and minimum value of the i-th objective in the population. After the ranking operation, the population is expected to be ranked from best solutions (class F_1) in a decreasing order of importance.

The crowd_dist(F_j) calculates a metric value of a solution in F_j providing an estimate of the neighboring members of F_j in both the objective and the decision variable space. The following pseudo-code describes the procedure:

```
crowd_dist($F_j$)
begin
   // initialize all distances to zero
   for $i = 1$ to |$F_j$|
      crowd_dist_obj($i$) = 0
      crowd_dist_var($i$) = 0
   // objective space crowding
   for $m = 1$ to $M$
      for $i = 1$ to |$F_j$|
         if $i$ is a minimum solution in $m$-th objective
              crowd_dist_obj($i$) = $\infty$
         else crowd_dist_obj($i$) += normalized_obj($i$)
   // decision variable space crowding
   for $j = 1$ to $n$
      for $i = 1$ to |$F_j$|
         if $i$ is a boundary solution in $j$-th variable
              crowd_dist_var($i$) += $2\times$normalized_var($i$)
         else crowd_dist_var($i$) += normalized_var($i$)
   // normalize distances and compute population average
   for $i = 1$ to |$F_j$|
      crowd_dist_obj($i$) = crowd_dist_obj($i$)/$M$
      crowd_dist_var($i$) = crowd_dist_var($i$)/$n$
   avg_crowd_dist_obj = $\sum_{i=1:|F_j|}$(crowd_dist_obj($i$))/|$F_j$|
   avg_crowd_dist_var = $\sum_{i=1:|F_j|}$(crowd_dist_var($i$))/|$F_j$|
   // if above average, assign larger of the two distances,
   // else assign smaller of the two distances
   for $i = 1$ to |$F_j$|
      if crowd_dist_obj($i$) > avg_crowd_dist_obj or
         crowd_dist_var($i$) > avg_crowd_dist_var
         crowd_dist($i$) =
           max(crowd_dist_obj($i$),crowd_dist_var($i$))
      else crowd_dist($i$) =
           min(crowd_dist_obj($i$),crowd_dist_var($i$))
end
```

To calculate the normalized_obj(i) of a solution i for the m-th objective, we first sort the population members in increasing order of the objective value and then calculate

$$\texttt{normalized\_obj}(i) = \frac{f_m(\texttt{right of } i) - f_m(\texttt{left of } i)}{f_m^{\max} - f_m^{\min}} \tag{3}$$

Similarly, the `normalized_var`(i) for the j-th variable is computed as follows:

$$\texttt{normalized\_var}(i) = \frac{x_j(\texttt{right of } i) - x_j(\texttt{left of } i)}{x_j^{\max} - x_j^{\min}} \tag{4}$$

Figure 2 illustrates the computation of objective space and decision variable space crowding distance. Note that if a solution is a boundary solution in the objective space, an infinite distance is assigned so as not to lose the solution. On the other hand, if a solution is a boundary solution in the decision variable space, the numerator in Equation 4 can be replaced by $(x_j(\texttt{right of } i) - x_j(i))$ or $(x_j(i) - x_j(\texttt{left of } i))$, as the case may be. Since this computes only a one-sided difference, the quantity is multiplied by a factor of two.

The `sorting(A)` sorts the population members of `A` into a decreasing order of crowding distance measure. Thus, the top-most member of the ordering has the maximum distance measure and the bottom-most member of the ordering has the least distance measure.

3.1 Computational Complexity

The `ranking` procedure involves $O(MN^2)$ computations, although a faster implementation can be achieved in $O(N \log^{M-2} N)$ computations [10, 9] for $M > 2$. The `crowd_dist` procedure involves $O(MN \log N)$ computations for calculating `crowd_dist_obj` values and $O(nN \log N)$ computations for calculating `crowd_dist_var` values. Thus, an efficient implementation of the algorithm requires an iteration-wise complexity of $O(N \log^{M-2} N)$ or $O(nN \log N)$, whichever is larger.

3.2 Single-Objective, Uni-global Optimizer

Now, let us analyze how the proposed omni-optimizer degenerates to different optimization algorithms of importance. First, we consider solving a single-objective minimization problem having one global optimum.

Here, the number of objectives is one, or $M = 1$. The tournament selection procedure described above degenerates to a constrained-tournament selection operator proposed in [3]. A check for dominance between two solutions degenerates to finding if a solution is better than the other or not. In the absence of constraints, the above procedure reduces to choosing the better of the two, and in case of a tie, the more diverse solution is chosen.

After the offspring population is created, it is combined with the parent population P_t, as it is done in standard single-objective EAs, such as in CHC [7] or in $(\mu+\lambda)$-ES [13]. The ranking procedure with a small ϵ parameter will assign each population member into a different class and the selection of N members from the combined population of size $2N$ degenerates to choosing the best N solutions of the combined parent-offspring population, a procedure followed in both CHC and $(\mu+\lambda)$-evolution strategies. If each class has only one or two members, the crowding distance computation assigns an infinite distance to them, thereby making no effect of crowding distance to the optimization procedure at all. However, in the presence of more than two solutions in a class which becomes the last class (class L) to be partially accepted, the extreme and sparsed solutions in the variable space are preferred. But we argue that such a decision, even though a low probability event, does not make much of a difference in obtaining the sole optimum of the problem.

Thus, if the underlying problem is a single-objective problem with one global optimum, the procedure is very similar to a standard EA procedure which uses tournament selection, genetic variation operators and an elite-preservation strategy along with a diversity preserving operator.

3.3 Single-Objective, Multi-global Optimizer

Often, there exist problems which have more than one global minimum solutions. The proposed omni-optimizer degenerates to finding multiple such global minima in a single simulation run.

The procedure becomes similar to the previous case, except that now there may exist multiple solutions in the top class (F_1), even towards the end of a simulation when the proposed method captures multiple optimum solutions. In such cases, the ranking procedure will put all such solutions in one class and the `crowd_dist` operation degenerates to a variable space crowding alone (since all these solutions will have identical function value or function values with a maximum difference of ϵ). In EA literature, such a variable space crowding operator emphasizing distant solutions in the variable space to remain in the population (often called a *niching* procedure) [8, 12] is used for finding multiple global optimum solutions. It is interesting to note how the proposed omni-optimization procedure degenerates to such a niching strategy automatically when there exist a number of global optimal solutions in a problem. Contrary to other niching methods, the proposed strategy does not require any additional parameter describing the niche size and it automatically adapts itself to find multiple optima offered by a problem.

3.4 Multi-objective Optimizer

In the presence of multiple objectives where each efficient solution in the objective space corresponds to a single Pareto-optimal solution in the decision space, the omni-optimization procedure degenerates to NSGA-II procedure [5] with the following modifications:

1. The ϵ-dominance criterion is used for classifying solutions into different fronts in the `ranking` procedure.
2. Both objective space and variable space crowding is performed for maintaining diversity among solutions of a single front.

The first modification makes the size of the non-dominated fronts larger than the non-dominated fronts obtained using the usual domination criterion. Since the selection begins from the top class and continues to worse classes, this modification does not make much of a difference in its working from that in NSGA-II. Due to the inclusion of variable-space niching in the omni-optimizer, the distribution of solutions is expected to be better in both objective and variable spaces.

However, a similar emphasis to that in NSGA-II has been followed in retaining extreme objective solutions, thereby ensuring a wide range of solutions in the objective space.

3.5 Multi-objective, Multi-global Optimizer

There exist some problems in which each efficient point in the objective space corresponds to a number of Pareto-optimal solutions in the decision variable space. In such problems, the task of an optimizer would be to find multiple such Pareto-optimal solutions corresponding to each efficient point. The proposed omni-optimizer can find such multiple solutions in a single simulation run.

The working of the proposed algorithm in such problems is similar to that in the previous case, except that in the `crowd_dist` operation, all such multiple solutions will be emphasized. These solutions will have identical objective function values, thereby making the `crowd_dist_obj` values to be zero (unless they are the extreme solutions), but their `crowd_dist_var` values will be non-zero. Since a solution's crowding distance value is chosen as the maximum of the two crowding distance values, these solutions will inherit the `crowd_dist_var` values. Thus, non-dominated solutions in a particular front can survive due to their sparsity either in the objective space or in the decision variable space. This allows not only sparsed efficient solutions to remain in the population, but any multiplicity of such efficient solutions in the decision variable space are also equally likely to survive in the population.

4 Simulation Results

In this section, we present simulation results of the omni-optimizer on various test problems chosen from the literature. In all problems, we use simulated binary crossover operator (with $\eta_c = 20$) and the polynomial mutation (with $\eta_m = 20$) [4] to handle real-valued variables. A crossover probability of 0.8 and a mutation probability of $1/n$ is used. For all problems, we use $\delta = 0.001$.

4.1 Single-Objective, Uni-global Test Problems

We choose 20-variable Rastrigin's function and 20-variable Schwefel's function. In both functions, there are many local minima, but there is only one global minimum and the corresponding function value is zero. Table 1 shows the best, median and worst number of evaluations needed in 10 runs of the proposed omni-optimizer to arrive at function value smaller than f^*. The seven-variable, four-constraint problem 5 described in [3] is solved next. With a population of size 70, the omni-optimizer is run for 5,000 generations and the best, median, and worst function values of 10 runs are found to be 680.216, 680.254, and 680.433, respectively. The best-known solution, as reported in [3], was $f(\mathbf{x}) = 680.630$ obtained for an identical number of function evaluations. For the welded beam design problem [3] having four variables and five non-linear inequality constraints, the omni-optimizer with a population size of 80 finds objective values of 2.381, 2.385,

Table 1. Results of omni-optimizer on single-objective, uni-global problems

Function	$f(\mathbf{x})$	n	Range	N	Target f^*	Func. Eval.		
						Best	Median	Worst
Rastrigin	$\sum_{i=1}^{n} x_i^2 + 10(1 - \cos(2\pi x_i))$	20	$[-10, 10]$	20	0.01	19,260	24,660	29,120
Schwefel	$418.9829n - \sum_{i=1}^{n} x_i \sin\sqrt{\lvert x_i \rvert}$	20	$[-500, 500]$	50	0.01	54,950	69,650	103,350

and 2.387 as best, median and worst of 10 runs of 4,000 generations each. The best result reported earlier with identical number of function evaluations also had a function value of 2.381.

4.2 Single-Objective, Multi-global Test Problems

We consider two test problems in this category. The first problem is a single-variable problem having 21 different global optimal solutions.

$$\text{Minimize} \quad f(x) = \sin^2(\pi x), \qquad x \in [0, 20]. \tag{5}$$

Here, we use a population of size 100. Figure 3 shows the obtained solutions after 200 generations. It is clear that all 21 global minimum solutions are found by the omni-optimizer.

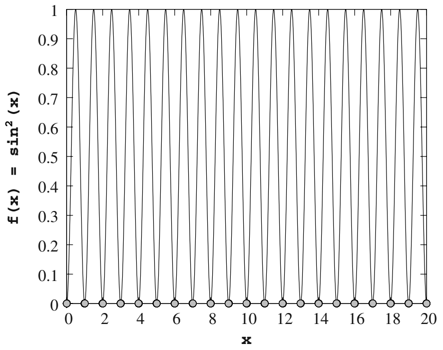

Fig. 3. All 21 minima are found for the $\sin^2(x)$ problem

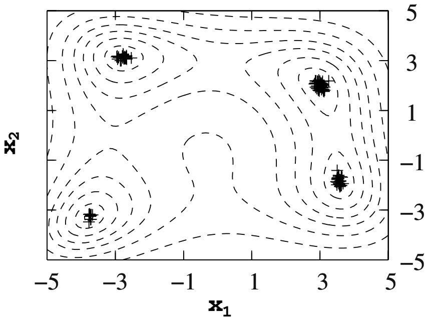

Fig. 4. All four minima are found for the Himmelblau's function

The second problem is the well-known Himmelblau's function [2]:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2, \quad -20 \le x_1, x_2 \le 20. \tag{6}$$

There are four minima, each having a function value equal to zero. Here, we use a population of size 100. Figure 4 shows the obtained solutions at the end of 100

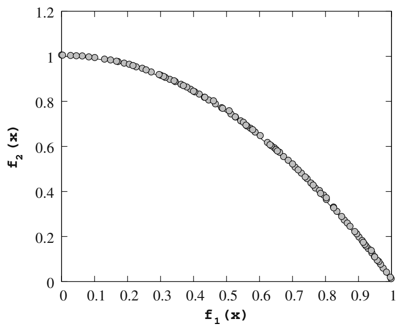

Fig. 5. Efficient points for ZDT2

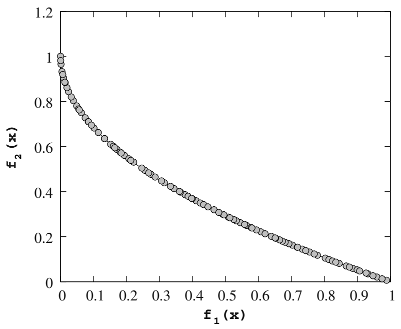

Fig. 6. Efficient points for ZDT4

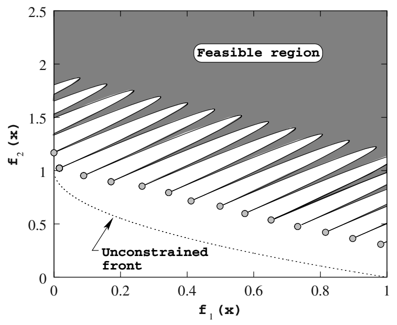

Fig. 7. Efficient points for CTP4

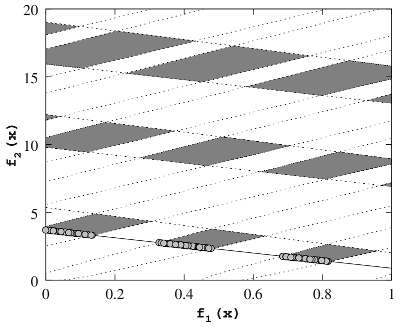

Fig. 8. Efficient points for CTP8

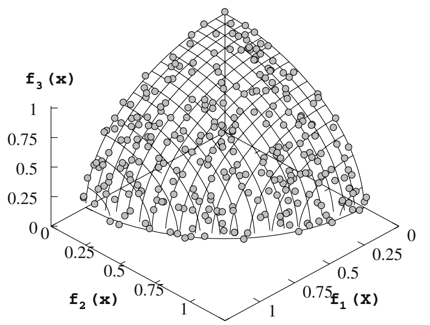

Fig. 9. Efficient points for DTLZ4S

generations. It is clear that the omni-optimizer is able to find all four minima in a single simulation run, similar to that reported in the literature [1] using a specialized niched EA.

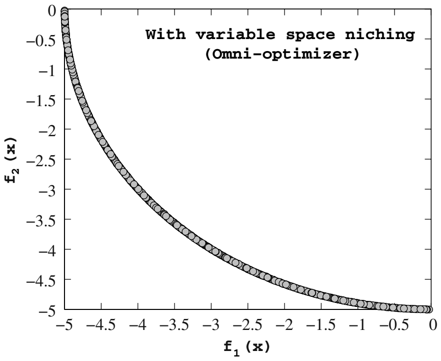

Fig. 10. Efficient points using omni-optimizer

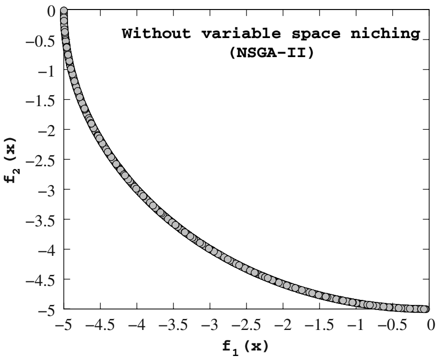

Fig. 11. Efficient points using NSGA-II

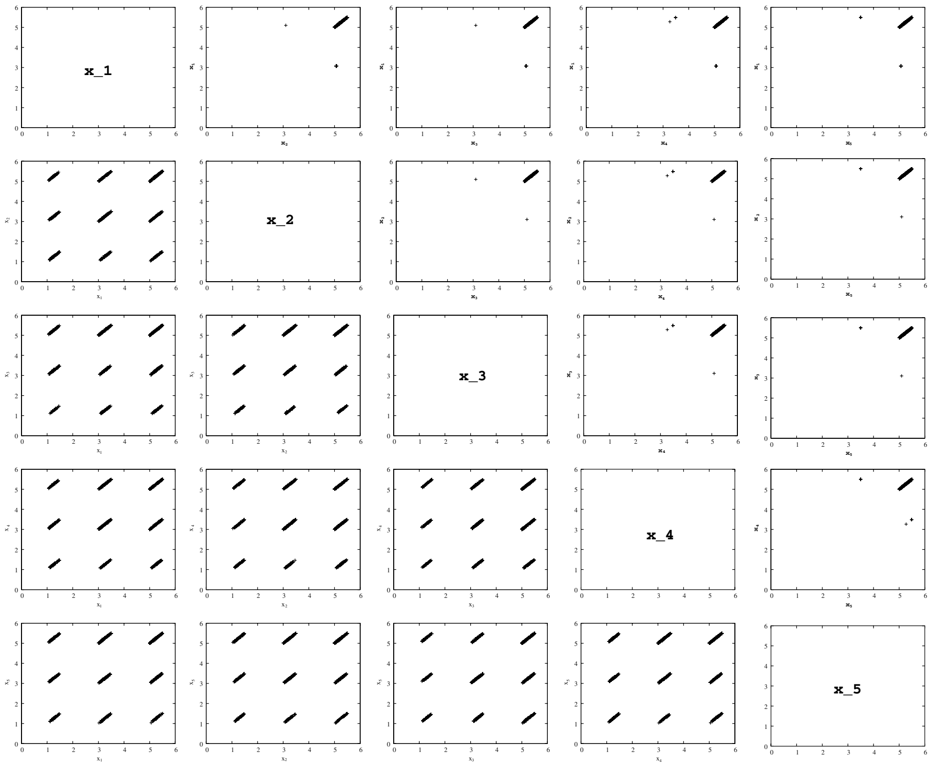

Fig. 12. Pareto-optimal solutions with omni-optimizer (left) and NSGA-II (right). The axes in a (i, j)-plot correspond to variables x_i and x_j

4.3 Multi-objective, Uni-global Test Problems

We choose unconstrained two-objective test problems – 30-variable ZDT2 and 10-variable ZDT4, and constrained problems CTP4 and CTP8 [4]. Also, we consider a three-objective test problem DTLZ4. For all problems, we use a population of size 100, except in DTLZ4 we use a population of size 300. Figures 5 till 9 show the corresponding efficient points obtained by the omni-optimizer. Due to different complexities of the problems, we have chosen to run till 100, 250, 7,000 and 100 generations, respectively, for ZDT2, ZDT4, CTP4 and CTP8. In the case of CTP4, the results are shown after 7,000 generations due to algorithm's sluggishness to reach the efficient points through the narrow feasible tunnels (see Figure 7).

4.4 Multi-objective, Multi-global Test Problem

We design the following problem:

$$\begin{aligned} \text{Minimize } f_1(x) &= \sum_{i=1}^{n} \sin(\pi x_i), \\ \text{Minimize } f_2(x) &= \sum_{i=1}^{n} \cos(\pi x_i), \\ 0 \le x_i &\le 6, \quad i = 1, 2, \ldots, n. \end{aligned} \tag{7}$$

Here, both objectives are periodic functions with period of 2. The efficient frontier corresponds to the Pareto-optimal solutions $x_i \in [2m+1, 2m+3/2]$, where m is an integer. We choose $n = 5$. For every efficient point in the objective space there are in general $3 \times 5! = 360$ Pareto-optimal solutions in the decision variable space. We choose a population of size 1,000 and run the algorithm for 500 generations to capture as many Pareto-optimal solutions as possible. It is interesting to note that both algorithms find the entire range of efficient points in the objective space, as shown in Figures 10 and 11. However, the variable space plots show a different scenario. The lower diagonal plots in Figure 12 show the performance of the omni-optimizer and the upper diagonal plots show that of the original NSGA-II. It is clear that multiple global solutions for different combinations of x_i variables are obtained using the omni-optimizer. Since no variable space crowding was considered in the original NSGA-II, not all global combinations are found.

5 Conclusions

The optimization literature deals with single and multi-objective optimization problems and uni-global and multi-global problems differently.

In this paper, for the first time, we present an omni-optimizer which is designed to solve different types of optimization problems usually encountered in practice: multi-objective, multi-global problems requiring to find multiple efficient solutions each corresponding to multiple Pareto-optimal solutions. The algorithm is designed in a way so that it degenerates to an efficient procedure for solving an optimization problem with a simpler task of finding a single global minimum in a single-objective optimization problem or multiple global minima in a single-objective optimization problem or multiple efficient solutions in a multi-objective optimization problem. Proof-of-principle simulation results on 11 different test problems and one welded-beam design problem indicate the efficacy of the proposed omni-optimizer, suggest immediate further evaluation of the procedure, and stress the importance of more such studies in the near future. More research is needed to compare the proposed approach with dedicated single and multi-objective optimization procedures and on more synthetic and real-world problems.

References

1. K. Deb. Genetic algorithms in multi-modal function optimization. Master's thesis, Tuscaloosa, AL: University of Alabama, 1989.
2. K. Deb. *Optimization for Engineering Design: Algorithms and Examples.* New Delhi: Prentice-Hall, 1995.
3. K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):311–338, 2000.
4. K. Deb. *Multi-objective optimization using evolutionary algorithms.* Chichester, UK: Wiley, 2001.
5. K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
6. M. Ehrgott. *Multicriteria Optimization.* Berlin: Springer, 2000.
7. L. J. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of Genetic Algorithms 1 (FOGA-1)*, pages 265–283, 1991.
8. D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 41–49, 1987.
9. M. T. Jensen. Reducing the run-time complexity of multiobjective EAs. *IEEE Transcations to Evolutionary Computation*, 7(5):503–515, 2003.
10. H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the Association for Computing Machinery*, 22(4):469–476, 1975.
11. K. Miettinen. *Nonlinear Multiobjective Optimization.* Kluwer, Boston, 1999.
12. A. Pétrowski. A clearing procedure as a niching method for genetic algorithms. In *IEEE 3rd International Conference on Evolutionary Computation (ICEC'96)*, pages 798–803, 1996.
13. H.-P. Schwefel. *Evolution and Optimum Seeking.* New York: Wiley, 1995.