Omni-optimizer: A Generic Evolutionary Algorithm for Global Optimization

Kalyanmoy Deb and Santosh Tiwari

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

Abstract. Due to vagaries of optimization problems encountered in practice, users resort to different algorithms for solving different optimization problems. In this paper, we suggest and evaluate an optimization procedure which specializes in solving a wide variety of engineering optimization problems. The proposed algorithm is designed as a generic multi-objective, multi-global optimizer. Care has been taken while designing the algorithm such that it automatically degenerates 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 chosen from literature. 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 of 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 engineering optimization problems often encountered

in practice. The proposed *omni-optimizer* algorithm adapts itself to solve different kinds of problems – single or multi-objective problems and uni-global or multi-global problems. The motivation for developing such a generic optimization procedure came from generic programming practices. Consider a very simple example: the programming task is to develop a code for adding 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.

A simpler version of the proposed omni-optimizer was found to solve a limited number of different optimization problems in an earlier study by the authors [6]. In this paper, we propose an extended omni-optimizer procedure which is carefully designed to have various properties needed for solving different kinds of optimization problems and is applied to a wide variety of existing and new test problems. In some very difficult optimization problems, the proposed omni-optimizer is also found to be computationally efficient than existing other EA implementations. The simulation results on various 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 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 in a single simulation run of the algorithm.

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

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

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 (Karush-Kuhn-Tucker) optimality conditions which involve finding the gradients of objective and constraint functions [2, 11, 15].

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 or all 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.

[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.

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, four individuals are chosen based on their mutual distance in genotypic space and two constrained binary-tournament selection operations are performed to select two parent solutions. We use a nearest neighbor based strategy to choose two individuals that take part in tournament selection. First individual is chosen randomly from the population and second individual is the one that is nearest to the first one in genotypic space. Such a strategy helps in preserving multi-global solutions and in general also speeds up the convergence rate. The tournament selection operator prefers feasible solutions over infeasible solutions (constraint handling [7]), 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. In the proposed algorithm, we use SBX crossover [8] and a slightly different variation of polynomial mutation [9] for real variables. For the case of binary variables, we use a standard two-point crossover and bitwise mutation. 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 not enough good offspring-solutions are 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 variable space, we select only those many solutions which will just 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 with added enhancements. 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) ∪ Shuffle(P_t)
        for i = 1 to N − 1 do
            // First selection operation
            (player1, player2) = choose_nearest (R_t)
            parent1 = tournament(player1, player2)
            //Second selection operation
            (player1, player2) = choose_nearest (R_t)
            parent2 = tournament(player1, player2)
            // crossover and mutation operators
    (offspring1, offspring2) = variation (parent1, parent2)
            Q_t(i)   = offspring1
            Q_t(i + 1) = offspring2
            i = i + 2   //increment iteration counter by 2
        R_t = P_t ∪ Q_t   // elite preservation
        (F_1, F_2, ...) = ranking(R_t) // best class F_1 and so on
        P_{t+1} = ∅
        j = 1  // class number
        while |P_{t+1} ∪ F_j| ≤ N
            P_{t+1} = P_{t+1} ∪ F_j  // include classes from best
            crowd_dist(F_j)  // assign crowding distance to 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} ∪ 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 `choose_nearest(`R_t`)` operator pops out two individuals from the shuffled population who take part in constrained binary tournament. Following pseudocode describes the `choose_nearest` procedure.

```
(player1, player2) = choose_nearest (R_t)
begin
    player1 = pop (R_t(1))
        dist = ∞
        for i=2 to size(R_t) do
           temp = compute_normalized_genotype_distance (player1, R_t(i))
        if (temp < dist)
```

```
            index = i
            dist = temp
        player2 = pop (R_t(index))
end
```

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:

$$\texttt{CV}(\texttt{a}) = \sum_{j=1}^{J} \langle \overline{g}_j(\texttt{a}) \rangle + \sum_{k=1}^{K} |\overline{h}_k(\texttt{a})|, \tag{2}$$

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

This is the usual *domination* principle used in multi-objective optimization [15, 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 (crossover followed by mutation) and creates two offspring solutions. We use the standard SBX crossover [8] with shuffling and 50% recombination. Such a recombination strategy facilitates exhaustive exploitation of the already explored search space. Here, we use a slightly modified formulation for polynomial mutation which is as follows:

$$\delta_1 = \frac{x_p - x_l}{x_u - x_l}, \tag{3}$$

$$\delta_2 = \frac{x_u - x_p}{x_u - x_l}, \tag{4}$$

$$\delta_q = \begin{cases} \left[2r + (1-2r)(1-\delta_1)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}} - 1 & \text{if} \quad r \leq 0.5, \\ 1 - \left[2(1-r) + 2(r-0.5)(1-\delta_2)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}} & \text{otherwise,} \end{cases} \tag{5}$$

$$x_c = x_p + \delta_q(x_u - x_l). \tag{6}$$

In the above equation,

- x_p is the parent solution (a real variable)
- x_c is the child solution (a real variable)
- x_l is the lower bound of the solution variable
- x_u is the upper bound of the solution variable
- r is a random number uniformly distributed in $[0, 1]$, and
- η_m is the index for polynomial mutation

In the actual polynomial mutation [9], δ was taken as $\min(\delta_1, \delta_2)$ was used instead of δ_1 & δ_2 in the formula for δ_q. The disadvantage of actual formulation is that, if a variable touches it's boundary, then mutation has no effect – which can lead to premature convergence. The proposed formulation prevents such an incident and on an average perturbs almost half of the solution variables away from the boundary that are lying on the boundary. The other advantage of this highly disruptive scheme is that it uses two different distributions in two different regions (x_l to x_p and x_p to x_u) of the search space, thereby giving a non-zero probability of generating a children in the entire search space for any location of the parent variable. Such a scheme prevents the solution from any bias that the search space or the problem may offer. This scheme specifically helps in problems ZDT4 [18] and 3-dimensional DTLZ4 [10]. The former problem offers a large number of local optima and later problem offers an entire boundary (that is part of efficient frontier surface) coinciding with a variable boundary.

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:

```
(F1, F2, ...) = ranking(Rt)
begin
   k = 1  // class counter
   repeat
      Fk = ∅
      for i = 1 to |Rt|
         for j = 1 to |Rt| and j ≠ i
             if Rt(j) ϵ-dominates Rt(i), break
         if (j = |Rt|)  // Rt(i) is ϵ-non-dominated
             Fk = Fk ∪ Rt(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 or fully 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 7. 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:

$$\epsilon_i = \delta(f_i^{\max} - f_i^{\min}), \tag{7}$$

where $f_i^{\max}$ and $f_i^{\min}$ are the maximum and minimum value of the i-th objective in the current combined population. After the ranking operation, the population is expected to be ranked from best solutions (class F_1) in a decreasing order of importance. Some salient features of the proposed ϵ-domination are as follows:

- It increases the size of the non-dominated set.
- It allows somewhat inferior solutions (depending on the value of δ) to remain in the population.
- It provides guaranteed diversity in case of single-objective optimization problems (for multi-objective problems, diversity is not an issue).
- It facilitates discovering multi-global solutions, especially in case of single objective problems.
- δ lies in the range $[0, 1]$. A value of $\delta = 0$ means usual domination and a value of $\delta = 1$ will result in entire population being non-dominated.

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
```

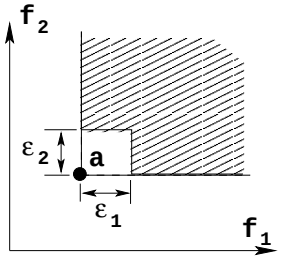


Fig. 1. The ϵ-dominance criterion. Point a ϵ-dominates the shaded region.

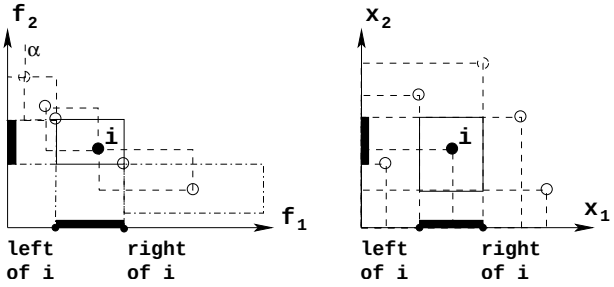


Fig. 2. Objective space and variable space crowding distance computations.

```
      for i = 1 to |F_j|
         if i is a minimum solution in m-th objective
              crowd_dist_obj(i) = ∞
         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×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 = ∑_{i=1:|F_j|}(crowd_dist_obj(i))/|F_j|
   avg_crowd_dist_var = ∑_{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

$$\texttt{normalized_obj}(i) = \frac{f_m(\texttt{right of } i) - f_m(\texttt{left of } i)}{f_m^{\max} - f_m^{\min}}. \quad (8)$$

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

$$\texttt{normalized_var}(i) = \frac{x_j(\texttt{right of } i) - x_j(\texttt{left of } i)}{x_j^{\max} - x_j^{\min}}. \quad (9)$$

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 9 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. In an actual implementation, extra processing is required to handle real and binary variables separately and in normalizing the objectives and variables taking care of boundary solutions.

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 as well as `choose_nearest` procedure involve $O(MN^2)$ computations. 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 $O(nN \log N)$ or $O(MN^2)$, 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.

```
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 f(a) < f(b), winner = a
        else if f(b) < f(a), winner = b
        else winner = (if crowd_dist(a) > crowd_dist(b)) ? a : b
end
```

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 [12] or in $(\mu + \lambda)$-ES [17]. 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 sparse 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 restricted 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) [13, 16] 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 any number of multiple optima offered by a problem.

3.4 Multi-objective Uni-global 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. Restricted Selection is used to choose two players that take part in the binary tournament.
2. The ϵ-domination criterion is used for classifying solutions into different fronts in the `ranking` procedure.
3. Both objective space and variable space crowding is performed for maintaining diversity among solutions of a single front.

The first modification in general speeds up the convergence (in this particular case). The second 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. The second modification may cause a somewhat degraded distribution in the objective space compared to 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. We also note that, usual domination checking is performed in tournament selection.

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 sparse 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 and polynomial mutation [4]. A crossover probability of 0.9 and a mutation probability of $1/n$ is used. For all problems, we use $\delta = 0.001$ unless otherwise mentioned. The value of crossover index η_c and mutation index η_m has been adjusted for different types of simulation runs. Typical range in which η_c and η_m is varied is $[1, 30]$. We have taken $\eta_c = 20$ and $\eta_m = 20$ unless otherwise specified.

4.1 Single-objective, Uni-global Test Problems

To evaluate the efficiency of omni-optimizer on single-objective test problems, we present the simulation results on a number of benchmark function gleaned from literature. We present the results for two kind of simulation studies. i) Number of function evaluations required to achieve a given threshold value, and ii) best optimum achieved in a given number of function evaluations. For the first kind, we choose 10 and 20 variable Rastrigin's and Schwefel's functions. 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 99 simulation runs of the proposed omni-optimizer to arrive at function value smaller than f^*.

Table 1. Results of omni-optimizer when a threshold value is prescribed.

Function	$f(\mathbf{x})$	n	Range	N	Target f^*	Func. Eval.		
						Best	Median	Worst
Rastrigin	$\sum_{i=1}^{n} x_i^2 + 10(1 - \cos(2\pi x_i))$	10	$[-10, 10]$	24	0.01	4,440	9,120	18,186
Rastrigin	$\sum_{i=1}^{n} x_i^2 + 10(1 - \cos(2\pi x_i))$	20	$[-10, 10]$	24	0.01	16,104	23,112	33,432
Schwefel	$418.9829n - \sum_{i=1}^{n} x_i \sin\sqrt{\lvert x_i\rvert}$	10	$[-500, 500]$	12	0.01	4,980	10,296	22,020
Schwefel	$418.9829n - \sum_{i=1}^{n} x_i \sin\sqrt{\lvert x_i\rvert}$	20	$[-500, 500]$	12	0.01	21,756	36,180	71,388

For the second case in which number of function evaluations is prescribed, we refer to Shinn *et al.* [14]. Simulation results of various algorithms have been

presented on 12 different test problems taken from literature. Problem definition, variable bounds and optimum value for all the functions is listed in Table 2. Simulation was done for a number of dimensions of variable space. Since in this particular case, our motive is to reach the optimum as fast as possible, we take very small values for crossover and mutation index. We take $\eta_c = 1$ and $\eta_m = 1$ for the purpose of performing simulations. Using such small values for η_c and η_m however hampers the performance of the algorithm in obtaining fine-grained fitness values. The average performance using 99 independent runs each for $D = 10$ to $D = 100$ (at steps of 10) are listed in Tables 3 and 4. We also present the best metric value reported by Shinn *et al.* [14] obtained by any algorithm in a particular problem. Maximum number of function evaluations specified is 10,000. We use the same performance metric $dist(D)$ [14] as used by the authors to report the outcome of omni-optimizer. It gives the mean distance between actual optimal solution $f_{opt}(D)$ and the best obtained solution $f_{best}(D)$ for one parameter as follows:

$$dist(D) = \frac{|f_{opt}(D) - f_{best}(D)|}{D} \tag{10}$$

Table 2. Problem definitions for the 12 benchmark functions

Definition of benchmark function	Variable bound	Optimum value
$f_1 = \sum_{i=1}^{D} \left[\sin(x_i) + \sin\left(\frac{2x_i}{3}\right)\right]$	[3, 13]	1.21598D (max)
$f_2 = -\sum_{i=1}^{D-1} \left[\sin(x_i + x_{i+1}) + \sin\left(\frac{2x_i x_{i+1}}{3}\right)\right]$	[3, 13]	$\approx$ 2D (max)
$f_3 = \sum_{i=1}^{D} \lfloor x_i + 0.5 \rfloor^2$	[-100, 100]	0 (min)
$f_4 = \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	[-5.12, 5.12]	0 (min)
$f_5 = \sum_{i=1}^{D} x_i^2$	[-5.12, 5.12]	0 (min)
$f_6 = \sum_{i=1}^{D} (x_i \sin(10\pi x_i))$	[-1.0, 2.0]	1.85D (max)
$f_7 = \sum_{i=1}^{D} \lvert\frac{\sin(10x_i\pi)}{10x_i\pi}\rvert$	[-0.5, 0.5]	0 (min)
$f_8 = 20 + e - 20e^{-0.2\sqrt{\frac{\sum_{i=1}^{D} x_i^2}{D}}} - e^{\sum_{i=1}^{D} \frac{\cos(2\pi x_i)}{D}}$	[-30, 30]	0 (min)
$f_9 = 418.9829D - \sum_{i=1}^{D} x_i \sin\left(\sqrt{\lvert x_i\rvert}\right)$	[-500, 500]	0 (min)
$f_{10} = \sum_{i=1}^{D-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]$	[-5.12, 5.12]	0 (min)
$f_{11} = 6D + \sum_{i=1}^{D} \lfloor x_i \rfloor$	[-5.12, 5.12]	0 (min)
$f_{12} = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	[-600, 600]	0 (min)

We notice that, omni-optimizer finds better solutions in all the cases (for all the dimensions of decision space) as compared to any of the algorithms reported by Shinn *et al.* [14]. It also shows that omni-optimizer is ideally suited for solving large scale parameter optimization problems and its performance is not hampered by increasing the dimension of decision space. A plot of $dist(D)$ versus D for all the 12 benchmark functions is shown in Figure 3.

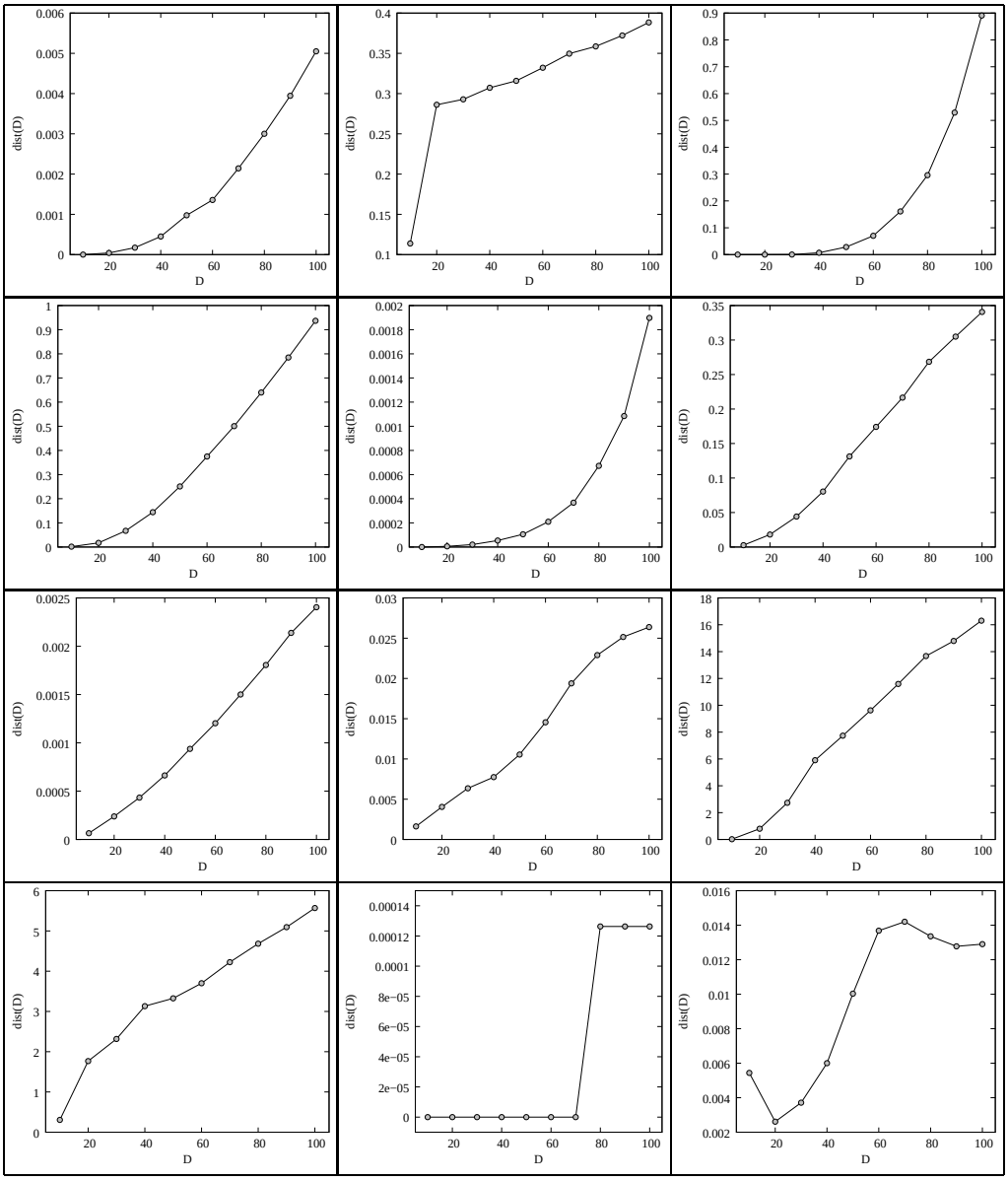


Fig. 3. Convergence-metric plots, first row comprises of functions f_1, f_2, f_3, second row comprises of functions f_4, f_5, f_6, and so on.

Table 3. Results of omni-optimizer for D=10 to 50 when number of function evaluations is prescribed.

Dimension / Function	Shinn et al. D = 10	10	20	30	40	50
f_1	1.22×10^{01}	0.00×10^{0}	4.00×10^{-5}	1.73×10^{-4}	4.50×10^{-4}	9.76×10^{-4}
f_2	1.65×10^{01}	1.14×10^{-1}	2.86×10^{-1}	2.93×10^{-1}	3.07×10^{-1}	3.16×10^{-1}
f_3	7.70×10^{-1}	0.00×10^{0}	0.00×10^{0}	3.37×10^{-4}	6.82×10^{-3}	2.83×10^{-2}
f_4	5.32×10^{00}	1.65×10^{-3}	1.71×10^{-2}	6.71×10^{-2}	1.44×10^{-1}	2.51×10^{-1}
f_5	3.00×10^{-4}	1.94×10^{-7}	5.56×10^{-6}	2.06×10^{-5}	5.40×10^{-5}	1.06×10^{-4}
f_6	1.80×10^{01}	2.61×10^{-3}	1.81×10^{-2}	4.42×10^{-2}	8.03×10^{-2}	1.31×10^{-1}
f_7	1.70×10^{-2}	6.46×10^{-5}	2.38×10^{-4}	4.33×10^{-4}	6.63×10^{-4}	9.38×10^{-4}
f_8	2.30×10^{-1}	1.62×10^{-3}	4.05×10^{-3}	6.35×10^{-3}	7.74×10^{-3}	1.05×10^{-2}
f_9	8.40×10^{00}	2.00×10^{-2}	7.98×10^{-1}	2.73×10^{0}	5.90×10^{0}	7.74×10^{0}
f_{10}	8.93×10^{00}	3.05×10^{-1}	1.77×10^{0}	2.32×10^{0}	3.13×10^{0}	3.33×10^{0}
f_{11}	3.00×10^{-2}	0.00×10^{0}	0.00×10^{0}	0.00×10^{0}	0.00×10^{0}	0.00×10^{0}
f_{12}	9.99×10^{-1}	5.44×10^{-4}	2.61×10^{-3}	3.71×10^{-3}	6.00×10^{-3}	1.00×10^{-2}

4.2 Single-Objective, Multi-Global Test Problems

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

$$\text{Minimize} \quad f(x) = \sin^2(\pi x), \qquad x \in [0, 20]. \tag{11}$$

Here, we use a population of size 100. Figure 4 shows the obtained solutions after 200 generations. It is clear that all 21 global optima are found by the omni-optimizer.

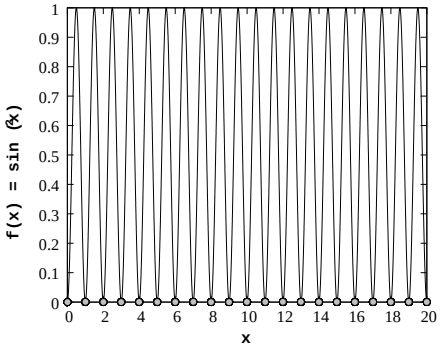


Fig. 4. All 21 minima are found for the $\sin^2(x)$ problem.

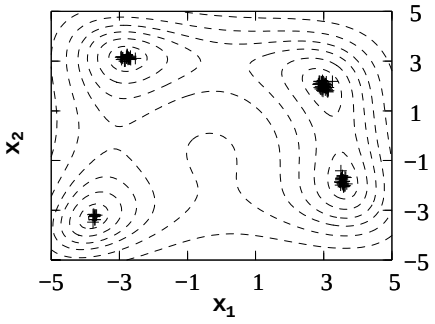


Fig. 5. All four minima are found for the Himmelblau's function.

Table 4. Results of omni-optimizer for D=60 to 100 when number of function evaluations is prescribed.

Dimension / Function	60	70	80	90	100	Shinn et al. D = 100
f_1	1.36×10^{-3}	2.14×10^{-3}	3.00×10^{-3}	3.95×10^{-3}	5.05×10^{-3}	1.20×10^{02}
f_2	3.32×10^{-1}	3.50×10^{-1}	3.59×10^{-1}	3.72×10^{-1}	3.88×10^{-1}	1.53×10^{02}
f_3	7.00×10^{-2}	1.61×10^{-1}	2.96×10^{-1}	5.29×10^{-1}	8.91×10^{-1}	6.21×10^{02}
f_4	3.75×10^{-1}	5.00×10^{-1}	6.40×10^{-1}	7.85×10^{-1}	9.37×10^{-1}	2.13×10^{02}
f_5	2.09×10^{-4}	3.67×10^{-4}	6.72×10^{-4}	1.08×10^{-3}	1.90×10^{-3}	1.60×10^{00}
f_6	1.74×10^{-1}	2.17×10^{-1}	2.68×10^{-1}	3.05×10^{-1}	3.41×10^{-1}	1.31×10^{02}
f_7	1.20×10^{-3}	1.50×10^{-3}	1.81×10^{-3}	2.14×10^{-3}	2.40×10^{-3}	6.50×10^{-1}
f_8	1.46×10^{-2}	1.94×10^{-2}	2.29×10^{-2}	2.52×10^{-2}	2.64×10^{-2}	3.69×10^{00}
f_9	9.62×10^{0}	1.16×10^{1}	1.37×10^{1}	1.48×10^{1}	1.63×10^{1}	8.01×10^{03}
f_{10}	3.70×10^{0}	4.22×10^{0}	4.68×10^{0}	5.09×10^{0}	5.57×10^{0}	2.08×10^{03}
f_{11}	0.00×10^{0}	0.00×10^{0}	1.26×10^{-4}	1.26×10^{-4}	1.26×10^{-4}	4.39×10^{01}
f_{12}	1.37×10^{-2}	1.42×10^{-2}	1.34×10^{-2}	1.28×10^{-2}	1.29×10^{-2}	3.29×10^{01}

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

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2, \quad -20 \leq x_1, x_2 \leq 20. \quad (12)$$

There are four minima, each having a function value equal to zero. Here, we use a population of size 100. Figure 5 shows the obtained solutions at the end of 100 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.

The third problem considered is the famous Weierstrass function.

$$\text{Minimize} f(x) = \sum_{i=1}^{D} \left[\sum_{k=0}^{k_{max}} \left[a^k cos(2\pi b^k x_i) \right] \right], \; x_i \in [-2, 2] \; \forall \; i \in \{1, ..., D\} \quad (13)$$

This problem is multi-modal and non-separable. We take $D = 2$, $a = 0.5$, $b = 3$ and $k_{max} = 20$ for the purpose of simulation run. There are 16 minima each having a function value equal to zero. The function attains a minimum when $x_i \in \{-1.5, -0.5, 0.5, 1.5\} \; \forall \; i \in \{1, ..., D\}$. Not all minima are found using $\delta = 0.001$, hence we relax the range further and choose $\delta = 0.05$ for this particular case. Using a large δ also degrades the obtained function value and therefore very fine grained fitness becomes difficult to obtain. We use a population of size 100 and perform 1000 generations. Figure 6 shows the obtained solutions at the end of simulation run.

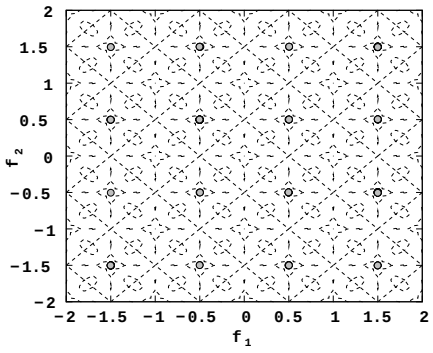


Fig. 6. All 16 minima are found for the weierstrass function

4.3 Multi-Objective, Uni-Global Test Problems

We choose unconstrained two-objective test problems – 10-variable ZDT4 & ZDT6 and 6-variable OSY and constrained problems 2-variable CTP4, CTP5 & CTP8 [4]. Also, we consider two three-objective test problems – 2-variable VNT and DTLZ4. For all problems, we use a population of size 100, except in OSY where we use a population size of 200 and DTLZ4 where we use a population of size 300. Figures 7 till 14 show the corresponding efficient points obtained by the omni-optimizer. Due to different complexities of the problems, we have chosen to run till 250, 250, 250, 7,000, 200, 100, 100 and 100 generations, respectively, for ZDT4, ZDT6, OSY, CTP4, CTP5, CTP8, VNT and DTLZ4.

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 10).

4.4 Multi-Objective, Multi-Global Test Problem

For demonstrating the performance of omni-optimizer on multi-objective, multi-global problems, we design two functions.

First function that we have designed is as follows:

$$\begin{aligned} \text{Minimize } f_1(x) &= \sum_{i=1}^{n} \sin(\pi x_i), \\ \text{Minimize } f_2(x) &= \sum_{i=1}^{n} \cos(\pi x_i), \\ 0 \le x_i &\le 6, \quad i = 1, 2, \ldots, n. \end{aligned} \tag{14}$$

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 15 and 16. However, the variable space plots show a different scenario. The lower diagonal plots in Figure 17 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.

The second function that we have designed is as follows:

$$\begin{aligned} \text{Minimize } f_1(x) &= \sin\left(\pi \sum_{i=1}^{n} x_i\right), \\ \text{Minimize } f_1(x) &= \cos\left(\pi \sum_{i=1}^{n} x_i\right), \\ 0 \le x_i \le 1, &\quad i = 1, 2, \ldots, n. \end{aligned} \tag{15}$$

Let $y = \sum_{i=1}^{n} x_i$, then range of y is $[0, 5]$ and the Pareto-optimal region corresponds to $1 \le y \le 1.5$ and $3 \le y \le 3.5$. This implies that any solution that gives the value of y in the Pareto-optimal region corresponds to Pareto-optimal front. Since the variables in consideration are continuous variables, there exist infinite solutions corresponding to every point on efficient frontier. In such a case, it is impossible to find all the Pareto-optimal solutions since they are too many, however an ideal algorithm should then uniformly distribute the population in the entire Pareto-optimal range for each variable. omni-optimizer does exactly the same thing and uniformly distributes the entire population on the Pareto-optimal hyper plane. It is noteworthy in this problem that though the entire search region constitutes the Pareto-optimal front, not all combinations of variables will give a solution on efficient frontier. This experiment demonstrates the flexibility and versatility of the proposed omni-optimizer. We run the algorithm with a population size of 500 for 500 generations. Figure 18 shows the variable space plots for both omni-optimizer and NSGA-II.

5 Conclusion

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 various test problems 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. K. Deb and S. Tiwari. Omni-optimizer: A procedure for single and multi-objective optimization. In *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO-2005), Lecture Notes on Computer Science 3410*, pages 41–65, 2005.
7. Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):311–338, 2000.
8. Kalyanmoy Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(2):115–148, 1995.
9. Kalyanmoy Deb and M. Goyal. A combined genetic adaptive search (geneas) for engineering design. *Computer Science and Informatics*, 26(4):30–45, 1996.
10. Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable Multi-Objective Optimization Test Problems. In *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 825–830, Piscataway, New Jersey, 2002. IEEE Service Center.
11. M. Ehrgott. *Multicriteria Optimization*. Berlin: Springer, 2000.
12. 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.
13. 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.
14. Shinn-Ying Ho, Li-Sun Shu, and Jian-Hung Chen. Intelligent evolutionary algorithms for large parameter optimization problems. *IEEE Transactions on Evolutionary Computation*, 8(6):522–541, 2004.
15. K. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer, Boston, 1999.
16. 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.

17. H.-P. Schwefel. *Evolution and Optimum Seeking*. New York: Wiley, 1995.
18. Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms : Empirical results. *Evolutionary Computation Journal*, 8(2):125–148, 2000.

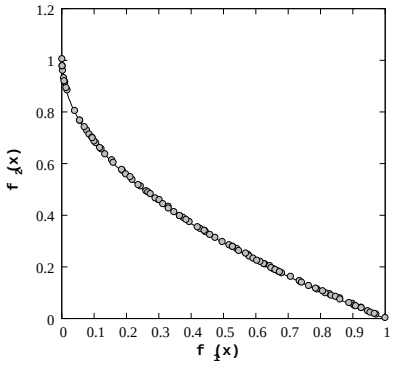


Fig. 7. Efficient points for ZDT4.

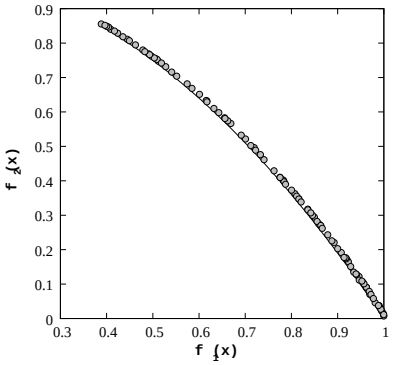


Fig. 8. Efficient points for ZDT6.

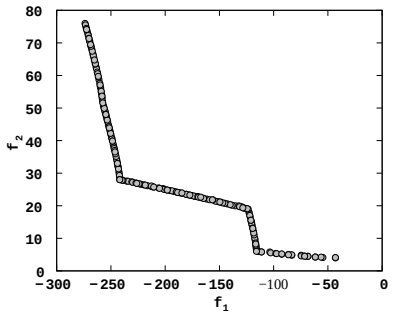


Fig. 9. Efficient points for OSY.

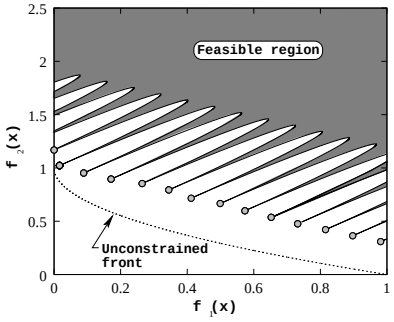


Fig. 10. Efficient points for CTP4.

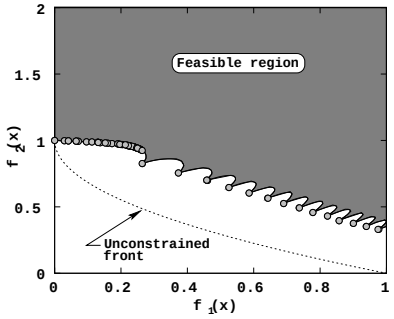


Fig. 11. Efficient points for CTP5.

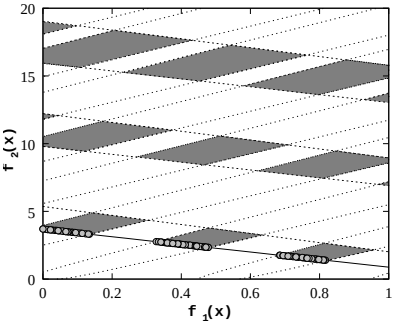


Fig. 12. Efficient points for CTP8.

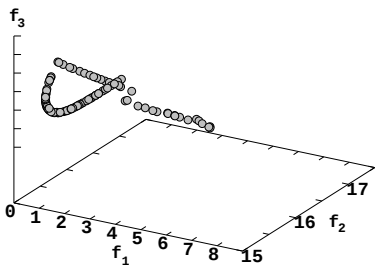


Fig. 13. Efficient points for VNT.

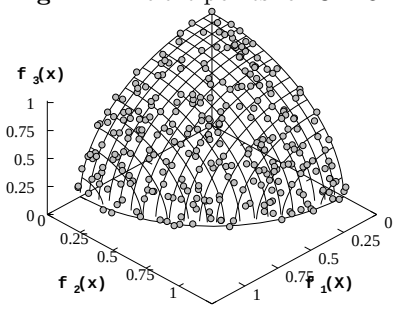


Fig. 14. Efficient points for DTLZ4.

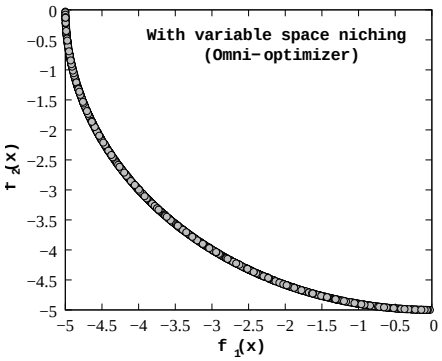


Fig. 15. Efficient points using omni-optimizer.

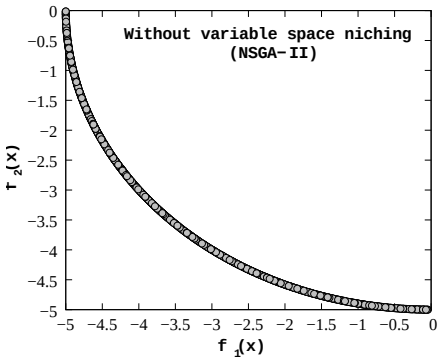


Fig. 16. Efficient points using NSGA-II.

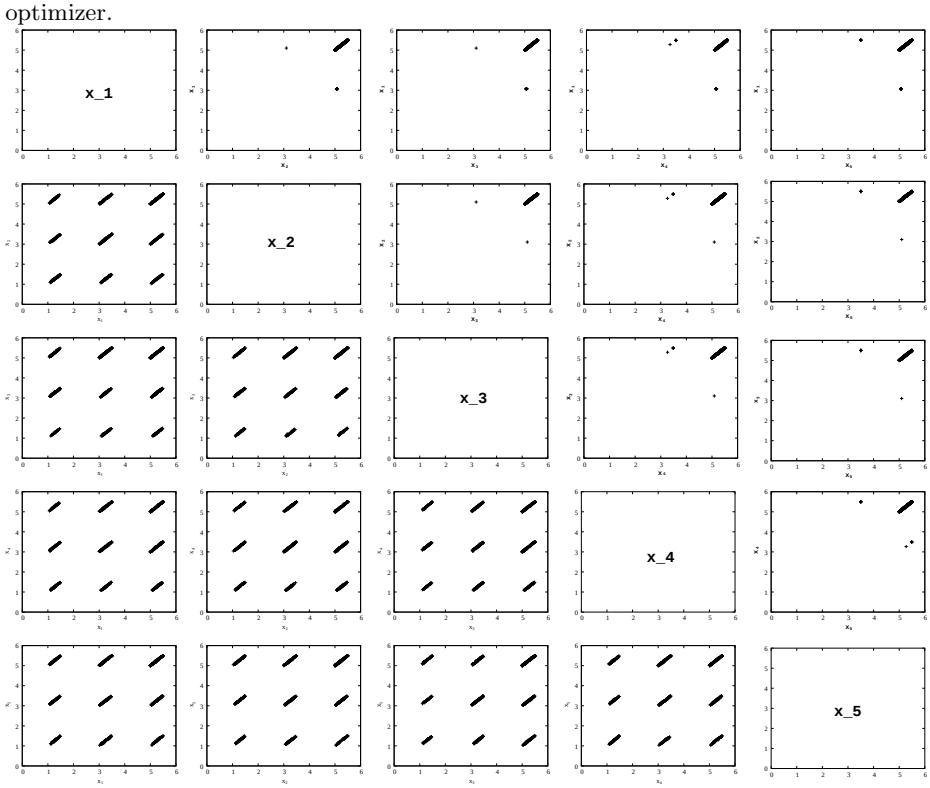


Fig. 17. Pareto-optimal solutions with omni-optimizer (left) and NSGA-II (right). The axes in a (i, j)-plot correspond to variables x_i and x_j.

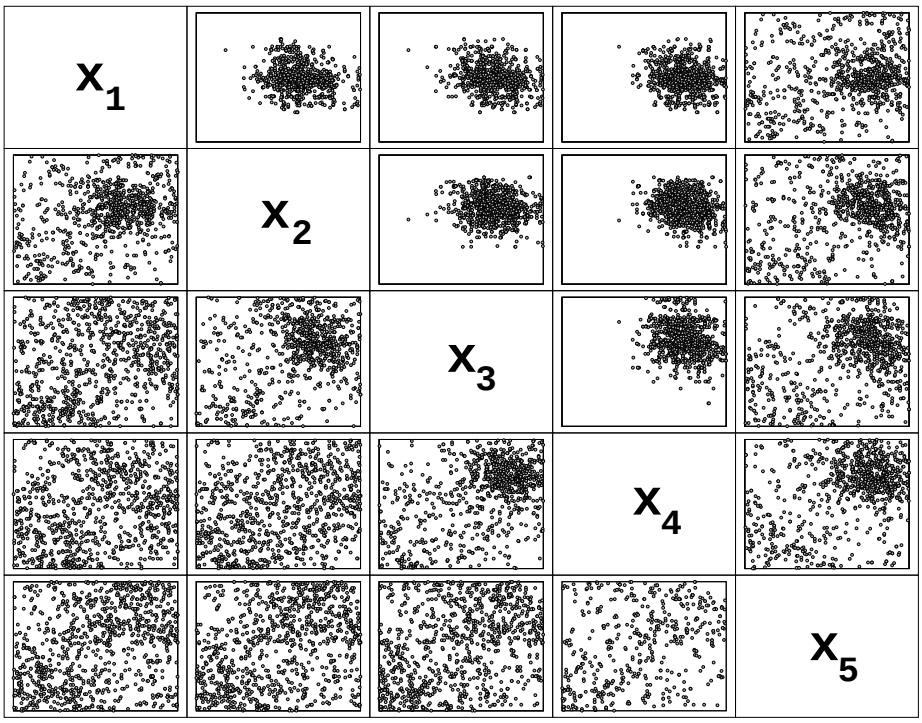


Fig. 18. Scatter plot of decision variables with omni-optimizer (left) and NSGA-II (right). The axes in a (i, j)-plot correspond to variables x_i and x_j.