Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 185 (2008) 1062–1087

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization

Kalyanmoy Deb [a,*], Santosh Tiwari [b]

[a] *Department of Mechanical Engineering, Indian Institute of Technology Kanpur, Kanpur 208 016, India*
[b] *Department of Mechanical Engineering, Clemson University, Clemson, SC 29634, USA*

Received 1 July 2005; accepted 1 June 2006
Available online 16 October 2006

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 and evaluate an optimization procedure which specializes in solving a wide variety of optimization problems. The proposed algorithm is designed as a generic multi-objective, multi-optima 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-optimal problems, single-objective multi-optima problems and multi-objective uni-optimal problems. The efficacy of the proposed algorithm in solving various problems is demonstrated on a number of test problems chosen from the literature. Because of its efficiency in handling different types of problems with equal ease, this algorithm should find increasing use in real-world optimization problems.

Keywords: Optimization; Multi-objective optimization; Pareto-optimal solutions; Niching; Constrained optimization; Evolutionary optimization

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 the 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

* Corresponding author. Tel.: +91 512 259 7205; fax: +91 512 259 7408.
E-mail addresses: deb@iitk.ac.in (K. Deb), stiwari@clemson.edu (S. Tiwari).

0377-2217/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2006.06.042

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 procedure for solving different kinds of 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-optimal or multi-optima problems. The motivation for developing such a generic optimization procedure comes from the generic programming practices, in which the purpose is often to develop an algorithm for an arbitrary value or number of input parameters. Although the algorithm is supposed to work well for any value, a good algorithm should also work well for specific values which may cause an internal degeneracy and may result into a simple trivial procedure. Viewing an optimization procedure along this spirit of programming practices, it is desirable to have an algorithm which is capable of handling any number of conflicting objectives, constraints, and variables. When the same algorithm is tried to solve a single-objective optimization problem, the procedure should degenerate to become a single-objective optimizer. Moreover, the proposed approach of this study can find multiple optimal solutions, if present in a problem, and automatically degenerates to find the sole optimum of a uni-optimal optimization problem. Proof-of-concept simulation results on a wide variety of test problems are provided that demonstrate the efficacy of the proposed algorithm on various types of problems.

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 [1]. 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 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 some of the other existing 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.

In Section 2, we present a brief description of various types of function optimization problems that algorithm attempts to solve. In Section 3, a detailed description of the proposed algorithm (along with pseudocodes) is presented. In Section 4, proof-of-concept simulation results on a wide variety of test problems are presented. In Section 5, we discuss some of the limitations of the proposed algorithm and discuss the possible future improvements to the algorithm. Section 6 presents a conclusion of the study.

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 sole optimum of the function. Other problems may contain more than one global optima in the search space, thereby requiring the task of finding multiple such globally 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 on finding the global optima (one or more) of one or more objective functions in a single simulation run of the algorithm.

Without loss of generality, we consider the following constrained M-objective ($M \geqslant 1$) minimization problem:

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

A n-variable solution vector $\mathbf{x}$ which satisfies all constraints and variable bounds as mentioned above is called a *feasible* solution. Mathematically, 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 [2–4]. For KKT optimality conditions to be applicable, the objective and constraint functions need to satisfy a set of criteria like convexity of objective and constraint functions and boundedness (from below for the case of minimization) of objective function. In case of a single objective problem, the optimum solution is the one which is feasible and have the minimum function value (for the case of minimization). For the case of multi-objective problems, the concept of Pareto-optimality is used, which is defined as follows [5]: *A point* $\mathbf{x}^*$ is Pareto-optimal *if for every feasible* $\mathbf{x}$ *and* $I = \{1, 2, \ldots, M\}$,

$$\forall_{i \in I}(f_i(\mathbf{x}^*) \leqslant f_i(\mathbf{x})) \tag{2}$$

and there exist at least one $i \in I$ *such that*

$$f_i(\mathbf{x}^*) < f_i(\mathbf{x}). \tag{3}$$

The corresponding objective vector $\mathbf{z} = (f_1(\mathbf{x}^*), f_2(\mathbf{x}^*), \ldots, f_M(\mathbf{x}^*))$ is called the *efficient objective vector*. Traditionally, the optimization literature has treated single and multi-objective problems separately. In the past, several evolutionary optimization algorithms specifically aimed at solving single and multi-objective optimization have been proposed and successfully applied [6–14]. All these algorithms specialize in solving either single or multi-objective problems (but not both). For handling multi-modal problems, techniques based on niching approaches [15,16] are employed. Unfortunately, every niching procedure demands a user-defined parameter (often called the niching parameter) and the performance of the algorithm is found to depend on the parameter value.

In this paper, for the first time, we suggest an omni-optimizer, in which a single optimization algorithm attempts to find 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-optimal optimization problems (the outcome is a single optimum solution),
2. Single-objective, multi-optima optimization problems (the outcome is multiple optimal solutions having the same function value),
3. Multi-objective, uni-optimal optimization problems (the outcome is multiple efficient points each corresponding to a single Pareto-optimal front), and
4. Multi-objective, multi-optima optimization problems (the outcome is multiple efficient points some or all of which may correspond to multiple Pareto-optimal fronts).

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 used 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 described above. It should also be noted that the algorithm attempts to find only the globally optimal solutions and has no mechanism to find locally optimal solutions and therefore cannot be applied to a problem for which local optima is desired.

3. Omni-optimizer

The proposed algorithm is an evolutionary optimization algorithm and comes under the category of generational genetic algorithms (GAs) [17]. It relies on genetic operators (such as crossover and mutation) to generate new solutions and imitates the natural process of evolution. The algorithm uses a two-tier fitness assignment scheme in which the primary fitness is computed using the *phenotypes* (objectives and constraint values). In contrast to most evolutionary algorithms (EAs), the secondary fitness is computed using both phenotypes and *genotypes* (decision variables). In EA's terminology, a solution is also referred to as an individual and a set of solutions is called a population [18,19]. Traditionally and in a binary-coded genetic algorithm (GA), variables are encoded in the form of a bit-string and the genetic operators are designed to operate directly on a population of bit-strings. However, in the recent years, real-parameter GAs are developed to work on real-valued variables using a number of recombination operators (such as SBX [20], PCX [8], UNDX [21], and others) and mutation operators (polynomial mutation, non-uniform mutation, and others [22]). The proposed omni-optimizer has been designed to handle both real-valued (continuous) and discrete-valued variables with equal ease using both binary-coded and real-parameter variation operators.

The optimization algorithm starts with an initial population P_0 of size N and an iteration counter t is initialized to zero. The initial population can be either generated randomly or preassigned. Omni-optimizer makes use of Latin-hypercube [23] based uniform sampling to generate the initial population. 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 (we call it restricted selection). First, an 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 favors competition between similar (in genotypic space) individuals. It helps in preserving multi-optimal solutions and in general also speeds up the convergence rate of the algorithm. The tournament selection operator prefers feasible solutions over infeasible solutions (for constraint handling [24]), non-dominated solutions over dominated solutions (for handling multiple objectives) and less-crowded solutions over more-crowded solutions (for 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 [20] and a slightly different variation of polynomial mutation [25] 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 [26]) 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. Iteration of the omni-optimizer is continued unless a termination criterion (like maximum number of generations) is met.

Readers familiar with the elitist non-dominated sorting GA (NSGA-II) [27] may find the proposed omni-optimization procedure similar to that of NSGA-II with some differences (the list of differences has been discussed after a detailed description of the algorithm). Thus, the proposed procedure is expected to solve multi-objective optimization problems in a manner similar to NSGA-II with some improvements (which can be attributed to restricted selection and a more disruptive mutation operator). 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-optimal and multi-optima 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 $N - 1$ 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
       end of for
    $R_t = P_t \cup Q_t$ // elite preservation
    $(F_1, F_2, \ldots)$ = ranking($R_t$) // best class $F_1$ and so on
    $P_{t+1} = \emptyset$
    $j = 1$ // class number
    while $|P_{t+1} \cup F_j| \leqslant N$ do
       $P_{t+1} = P_{t+1} \cup F_j$ // include classes from best
       crowd_dist($F_j$) // assign crowding distance to each soln.
       $j = j + 1$
    end of while
    $L = j$ // last class to be included partially
    rem = $N - |P_{t+1}|$// remaining solutions to be filled
    // sort $F_L$ in decreasing order of crowding distance
    sorting(crowd_dist($F_L$))
    $P_{t+1} = P_{t+1} \cup F_L$ (1:rem) // include top solutions
    $t = t + 1$ // increment iteration counter
    until (termination)
end
```

We now discuss the components of the omni-optimizer.

3.1. Selection operator with constraint handling approach

The operator shuffle(P_t) makes a random ordering of the population members of P_t. The choose_nearest(R_t) operator pops out a solution from the stack R_t and chooses another solution from R_t which is closest to the first solution. These two solutions then take part in constrained binary tournament selection and the better of them is chosen. Following pseudocode describes the choose_nearest procedure.

```
(player1, player2) = choose_nearest ($R_t$)
begin
    player1 = pop ($R_t$(1))
    dist = $\infty$
    for $i = 2$ to size($R_t$) do
       temp = compute_normalized_genotype_distance (player1, $R_t$(i))
       if (temp < dist)
          index = i
          dist = temp
       end of if
    end of for
    player2 = pop ($R_t$(index))
end
```

The routine compute_normalized_genotypic_distance computes the Euclidean norm of two solutions in the decision space. Every variable is normalized (with respect to its minimum and maximum value in the current population) before the norm is computed. 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
```

There are several ways to handle constraints in an optimization problem. Both penalty-based [28] and penalty-parameter-less [24] approaches have been successfully applied with EA approaches. The penalty-parameter-less approach is ideally suited for optimization algorithms working with a population, since an EA only needs to know, given two solutions in a tournament selection operator, which is better. The above procedure declares a feasible solution to be always better than an infeasible solution, a feasible solution with a better domination or crowding level is better than another feasible solution, and an infeasible solution with a smaller normalized constraint violation is better than another infeasible solution. The function `CV(a)` calculates the overall normalized constraint violation of solution a, as follows:

$$\mathrm{CV}(\mathrm{a}) = \sum_{j=1}^{J} \langle \bar{g}_j(\mathrm{a}) \rangle + \sum_{k=1}^{K} |\bar{h}_k(\mathrm{a})|, \tag{4}$$

where $\langle \cdot \rangle$ is a bracket operator [2]. Every constraint is normalized before computing the constraint violation since the scale in which different constraints vary may be different. In the case of multiple objectives, a dominating feasible solution wins over a dominated feasible solution. In the event of both feasible solutions being non-dominated to each other, the one having a larger crowding distance measure wins. The procedure for computing the crowding distance metric `crowd_dist()` is discussed a little later. The `random_choose(a, b)` selects a or b randomly with 50% probability.

3.2. Variation operators

Next, we discuss the variation operators used in the omni-optimizer. The variation operator takes two solutions and performs genetic operations (a recombination followed by a mutation) and creates two offspring solutions. We use the standard variable-wise SBX crossover [20] on 50% of variables, on an average. For each variable, after the two new values are computed, they are randomly associated with any one of two offspring solutions. Such a recombination strategy facilitates exhaustive exploitation of the already explored search space. Here, we use a slightly modified formulation for the polynomial mutation, which is as follows:

$$\delta_1 = \frac{x_\mathrm{p} - x_\mathrm{l}}{x_\mathrm{u} - x_\mathrm{l}}, \tag{5}$$

$$\delta_2 = \frac{x_\mathrm{u} - x_\mathrm{p}}{x_\mathrm{u} - x_\mathrm{l}}, \tag{6}$$

$$\delta_q = \begin{cases} [2r + (1-2r)(1-\delta_1)^{\eta_\mathrm{m}+1}]^{\frac{1}{\eta_\mathrm{m}+1}} - 1, & \text{if} \quad r \leqslant 0.5, \\ 1 - [2(1-r) + 2(r-0.5)(1-\delta_2)^{\eta_\mathrm{m}+1}]^{\frac{1}{\eta_\mathrm{m}+1}}, & \text{otherwise}, \end{cases} \tag{7}$$

Thereafter, the mutated variable value x_c is computed as follows:

$$x_c = x_p + \delta_q(x_u - x_l). \tag{8}$$

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 original polynomial mutation operator [22], $\delta = \min(\delta_1, \delta_2)$ was used in place of δ_1 and δ_2 in Eq. (7). The disadvantage of the original formulation is that, if a variable reaches its boundary, then mutation has no effect. This can lead to a premature convergence. The proposed formulation uses two different probability distributions in two different regions (x_l to x_p and x_p to x_u) of the search space, thereby assigning a non-zero probability of generating an offspring in the entire search space, even if one of the parent variable is on its boundary. The large ergodicity present in the mutation operator provides additional diversity which may be necessary in handling multi-modal problems.

3.3. Non-dominated ranking approach

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

```
(F_1, F_2,...) = ranking(R_t)
begin
  $k = 1$ // class counter
  repeat
    $F_k = \emptyset$
    for $i$ = 1 to |R_t| do
      for $j$ = 1 to |R_t| and $j \neq i$ do
      if R_t(j) $\epsilon$-dominates R_t(i), break
    end of for
    if ($j = |R_t|$) // R_t(i) is $\epsilon$-non-dominated
      $F_k = F_k \cup R_t(i)$
    end of if
  end of for
  $R_t = R_t \setminus F_k$
  $k = k + 1$
  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.

3.4. ϵ-Dominance operation

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

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

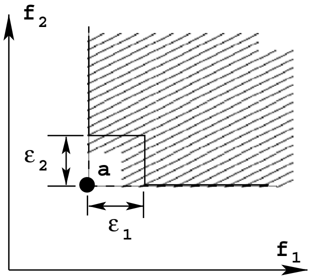

Fig. 1. The ϵ-dominance criterion. Point a ϵ-dominates the shaded region.

Here, ϵ_i is a quantity calculated from a user-defined parameter ϵ, described in Eq. (9). Fig. 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 ϵ as follows:

$$\epsilon_i = \epsilon(f_i^{\max} - f_i^{\min}), \tag{9}$$

where $f_i^{\max}$ and $f_i^{\min}$ are the maximum and minimum value of the ith 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 ϵ) 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-optima solutions, especially in case of single objective problems.
- ϵ lies in the range [0, 1]. A value of $\epsilon = 0$ means usual domination and a value of $\epsilon = 1$ will result in the entire population being non-dominated.

3.5. Crowding distance operator

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
  end of for
  // 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($i$)
    end of for
  end of for
  // 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($i$)
      end of for
    end of for
    // 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($i$)/$n$
    end of for
    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 of for
  end
```

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

$$\texttt{normalized\_obj}(i) = \frac{f_m(\texttt{right of i}) - f_m(\texttt{left of i})}{f_m^{\max} - f_m^{\min}}. \tag{10}$$

Similarly, the `normalized_var`(i) for the jth variable is computed as follows:

$$\texttt{normalized\_var}(i) = \frac{x_j(\texttt{rightofi}) - x_j(\texttt{leftofi})}{x_j^{\max} - x_j^{\min}}. \tag{11}$$

Fig. 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 Eq. (11) can be replaced by (x_j(`right of i`) − $x_j(i)$) or ($x_j(i)$ − x_j(`left of i`)), as the case may be. Since

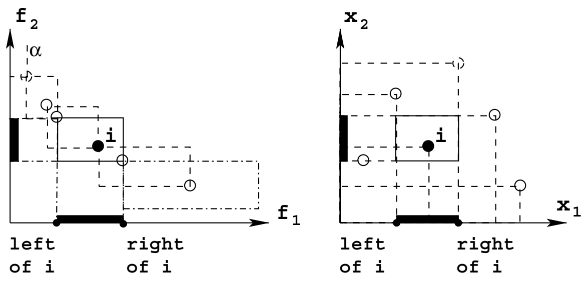

Fig. 2. Objective space and variable space crowding distance computations.

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 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.

3.6. 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 $O(nN\log N)$ or $O(MN^2)$, whichever is larger.

4. Omni-optimizer for different problem-solving tasks

Now, let us analyze how the proposed omni-optimizer degenerates to specialized algorithms for solving different types of optimization problems.

4.1. Single-objective, uni-optimal optimizer

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 [24]. 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 is done in standard single-objective EAs, such as in CHC [7] or in $(\mu + \lambda)$-ES [29]. The ranking procedure with a small ϵ 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 a restricted tournament selection, genetic variation operators and an elite-preservation strategy along with a diversity preserving operator.

4.2. Single-objective, multi-optima 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 the EA literature, such a variable space crowding operator emphasizing distant solutions in the variable space to remain in the population (called a *niching* procedure) [15,16] 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 (such as sharing, clearing, clustering, and others), 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.

4.3. Multi-objective, uni-optimal 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 the NSGA-II procedure [27] with the following modifications:

1. Restricted selection (instead of random selection) is used to choose two players that take part in the constrained binary tournament.
2. The ϵ-domination criterion is used for classifying solutions into different fronts in the `ranking` procedure. Tournament selection still uses usual domination for comparing solutions.
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. 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, a good distribution of solutions is expected 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.

4.4. Multi-objective, multi-optima optimizer

There exist some problems in which each efficient point in the objective space corresponds to more than one 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 (if the average value of variable and objective space crowding is more than the population average), these solutions will inherit the `crowd_dist_var` values if they are more diverse in the population. 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 likely to survive in the population.

5. 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 [22]. A crossover probability of 0.9 and a mutation probability of $1/n$ (n is the number of real variables) is used for all problems. For all problems, we use $\epsilon = 0.001$ (except for Weierstrass problem, for which we use $\epsilon = 0.05$). We have chosen $\eta_c = 1$ and $\eta_m = 1$ for all simulations. Here, we would like to emphasize the fact that fine-tuning the values of crossover and mutation index for every problem can yield significantly better results than the results reported in the paper. Here, we show its effect on multi-objective test problems.

5.1. Single-objective, uni-optimal test problems

To evaluate the efficiency of omni-optimizer on single-objective uni-optimal test problems, we present the simulation results on a number of benchmark function taken from the EA literature. We present the results for two performance criteria: (i) number of function evaluations required to achieve a given threshold value (f^*), 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 a function value smaller than f^*.

For the second case in which number of function evaluations is prescribed, we refer to Shinn et al. [30]. Simulation results of various algorithms have been presented on 12 different test problems taken from the literature. Problem definition, variable bounds and optimum value for all the functions are listed in Table 2. The

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

Function	$f(\mathbf{x})$	n	Range	N	Target f^*	Function evaluation		
						Best	Median	Worst
Rastrigin	$\sum_{i=1}^{n} x_i^2 + 10(1-\cos(2\pi x_i))$	10	[−10, 10]	40	0.01	8120	15,520	53,480
Rastrigin	$\sum_{i=1}^{n} x_i^2 + 10(1-\cos(2\pi x_i))$	20	[−10, 10]	40	0.01	24,520	41,760	106,440
Schwefel	$418.9829\, n - \sum_{i=1}^{n} x_i \sin\sqrt{\lvert x_i\rvert}$	10	[−500, 500]	16	0.01	6304	11,360	24,704
Schwefel	$418.9829n - \sum_{i=1}^{n} x_i \sin\sqrt{\lvert x_i\rvert}$	20	[−500, 500]	16	0.01	26,128	36,272	82,096

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]	≈2D (max)
$f_3 = \sum_{i=1}^{D}\lfloor x_i + 0.5\rfloor^2$	[−100, 100]	0 (min)
$f_4 = \sum_{i=1}^{D}[x_i^2 - 10\cos(2\pi x_i) + 10]$	[−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)
$f_8 = 20 + e - 20\mathrm{e}^{-0.2\sqrt{\frac{\sum_{i=1}^{D} x_i^2}{D}}} - \mathrm{e}^{\sum_{i=1}^{D}\frac{\cos(2\pi x_i)}{D}}$	[−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)
$f_{10} = \sum_{i=1}^{D-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	[−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)

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

Function	Dimension					
	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}

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

Function	Dimension					
	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}

average performance using 99 independent runs each for $D = 10$ to $D = 100$ (at steps of 10) are listed in Tables 3 and 4. Let the best function value obtained in ith simulation be f_i, then average value over $N = 99$ simulations is given by $f^* = \sum_{i=1}^{N} f_i / N$. We also present the best metric value reported by Shinn et al. [30] obtained by the best performing algorithm for that problem. Maximum number of function evaluations specified is 10,000. We use the same performance metric dist(D) [30] as used by the authors to report the outcome of omni-optimizer. It provides the mean distance between actual optimal solution $f_{\text{opt}}(D)$ and the best obtained solution $f_{\text{best}}(D)$ for one parameter as follows:

$$\text{dist}(D) = \frac{|f_{\text{opt}}(D) - f_{\text{best}}(D)|}{D}. \tag{12}$$

We notice that, the omni-optimizer finds better solutions in all 10 to 100-variable problems, as compared to the results reported by Shinn et al. [30]. It also shows that the omni-optimizer is ideally suited for solving large-scale optimization problems and its performance does not degrade significantly by increasing the dimension of decision space. A plot of dist(D) versus D for all 12 benchmark functions is shown in Fig. 3. These plots are better than those reported in the previous study [30].

5.2. Single-objective, multi-optima test problems

We consider three test problems in this category. In these problems, the difficulty comes in finding all or many optimal solutions which each problem offers. The algorithm has been designed to preserve any solution

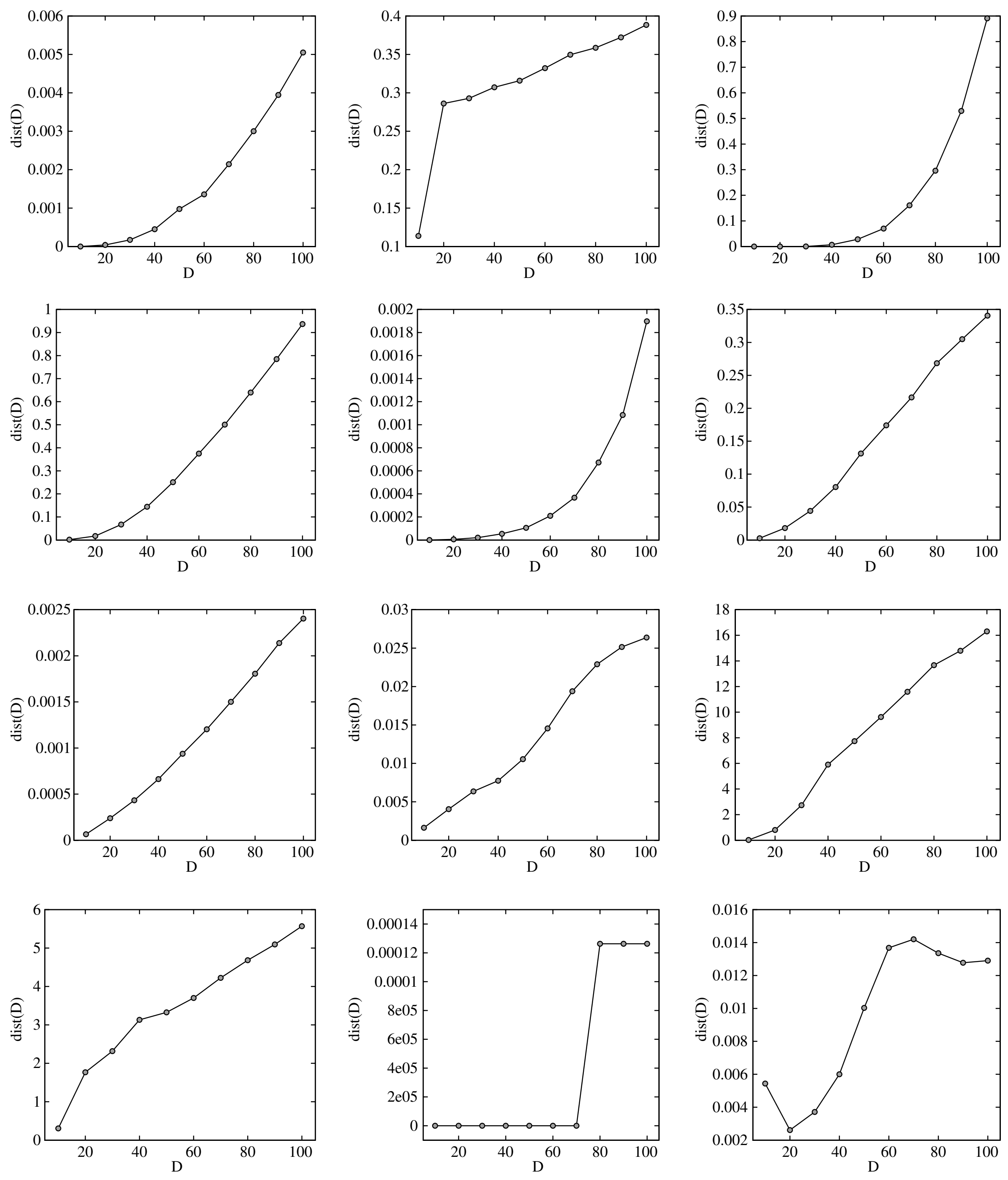

Fig. 3. Convergence-metric plots, first row comprises of functions f_1, f_2, f_3, second row comprises of functions f_4, f_5, f_6, and so on.

that is genetically diverse (as compared to other members of population) even if that solution has slightly worse function value. The initial stage of the evolution process is important since it determines if a population as a whole should move towards a specific point or should maintain a diversity in a particular region of the search space. The first problem is a single-variable problem having 21 different global optimal solutions

$$\text{Minimize } f(x) = \sin^2(\pi x), \qquad x \in [0, 20]. \tag{13}$$

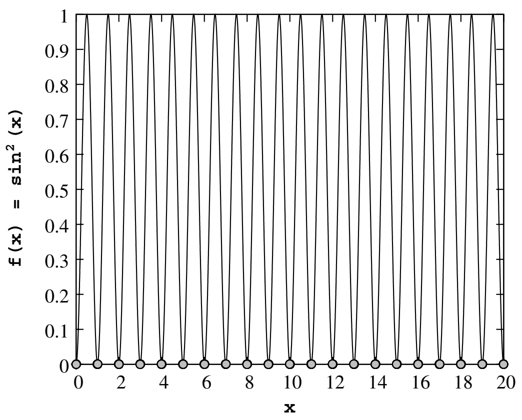

Fig. 4. All 21 minima are found for the $\sin^2(x)$ problem.

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

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

$$\text{Minimize } f(x_1,x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2, \quad -20 \leqslant x_1, x_2 \leqslant 20. \tag{14}$$

There are four minima, each having a function value equal to zero. Here, we use a population of size 100. Fig. 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 [31] using a specialized niched EA.

The third problem considered is the famous Weierstrass function:

$$\text{Minimize } f(x) = \sum_{i=1}^{D}\left[\sum_{k=0}^{k_{\max}}[a^k cos(2\pi b^k x_i)]\right], \quad x_i \in [-2,2] \quad \forall \quad i \in \{1,\ldots,D\}. \tag{15}$$

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 $x_i \in \{-1.5, -0.5, 0.5, 1.5\} \forall i \in \{1, \ldots, D\}$. Not all minima are found using $\epsilon = 0.001$, hence we relax the range further and choose $\epsilon = 0.05$ for this particular case. We use a population of size *100* and perform *1000* generations. All 16 minima are found by the omni-optimizer. A random sampling of solu-

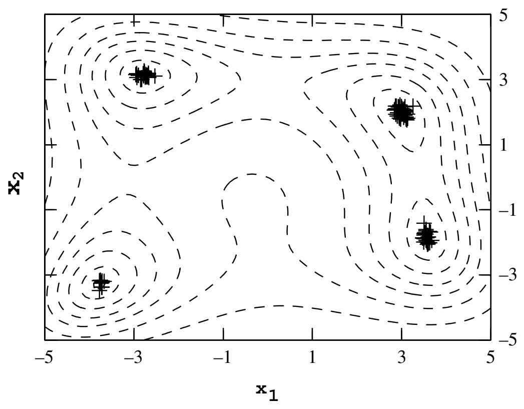

Fig. 5. All four minima are found for the Himmelblau's function.

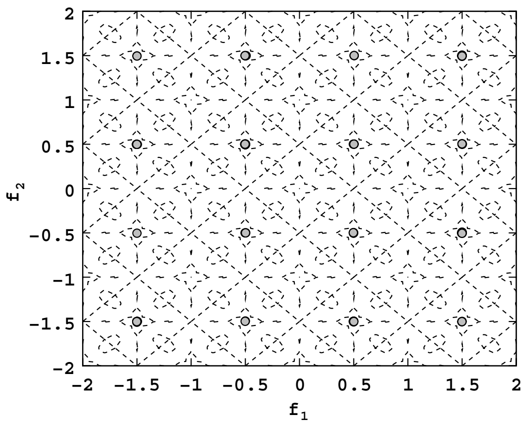

Fig. 6. All 6 minima are found for the Weierstrass function.

tions using *100,000* function evaluations yielded only two solutions with a value less than 0.1. Fig. 6 shows the obtained solutions at the end of simulation run.

5.3. Multi-objective, uni-global test problems

To demonstrate the efficiency of the algorithm on multi-objective test problems, we choose a number of two-objective test problems (10-variable ZDT4 and ZDT6 and 6-variable OSY, two-variable constrained problems CTP4, CTP5 and CTP8 [22]) and three-objective test problems (two-variable VNT [22] and 12-variable DTLZ4 [32]). For all these problems, we use a population of size 100, except in OSY and DTLZ4, where we use a population size of 200 and 300, respectively. Figs. 7–14 show the corresponding efficient objective vectors obtained by the omni-optimizer. Due to different complexities of the problems, we have chosen to run omni-optimizer till 250, 250, 250, 7000, 200, 100, 100 and 100 generations for ZDT4, ZDT6, OSY, CTP4, CTP5, CTP8, VNT and DTLZ4, respectively. In the case of CTP4, the results are shown after 7000 generations since all Pareto-optimal solutions are on a non-linear constraint boundary (see Fig. 10). Recently Deb et al. [26] proposed an epsilon-MOEA and compared its performance with NSGA-II, PESA, and SPEA2. The results were reported on five test problems including ZDT1, ZDT2, ZDT3, ZDT4 and ZDT6. Performance metrics considered were a convergence measure, a sparsity measure and the hyper-volume measure

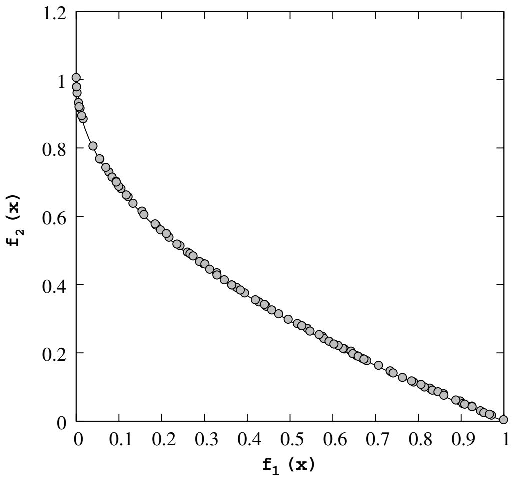

Fig. 7. Efficient points for ZDT4.

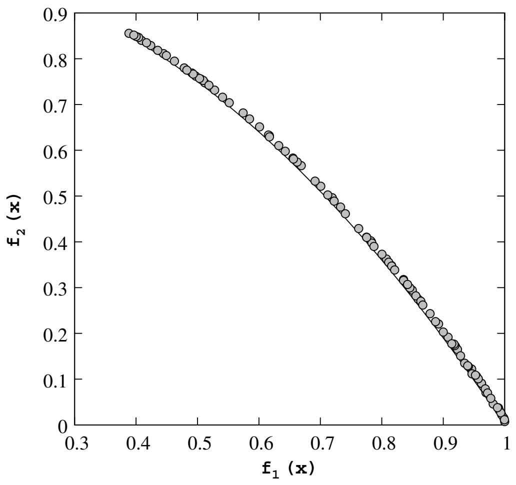

Fig. 8. Efficient points for ZDT6.

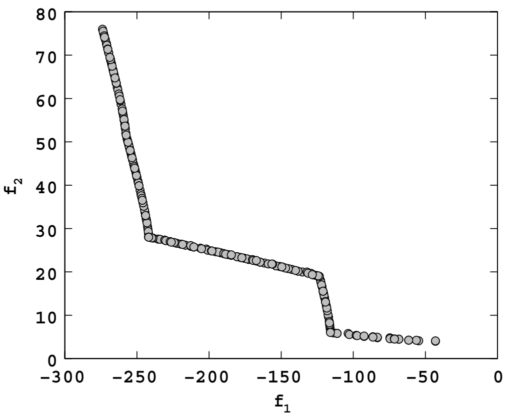

Fig. 9. Efficient points for OSY.

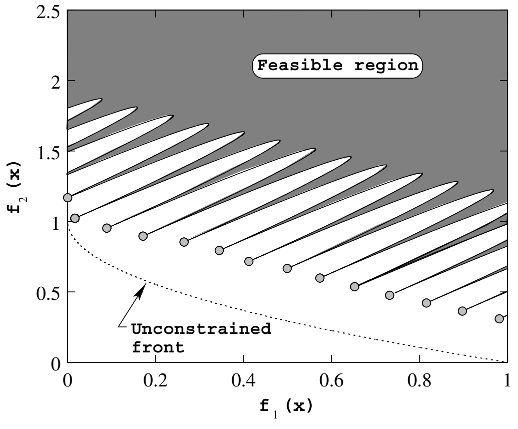

Fig. 10. Efficient points for CTP4.

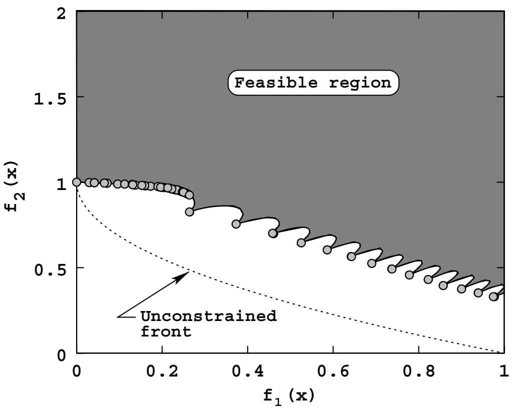

Fig. 11. Efficient points for CTP5.

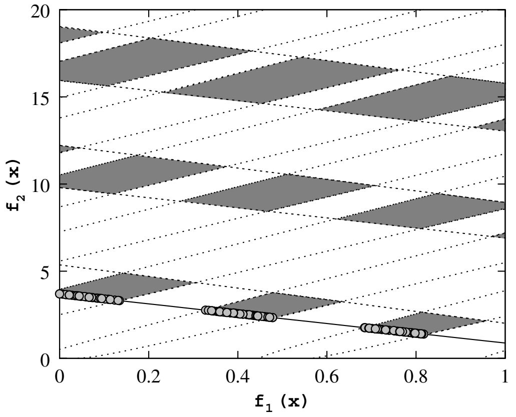

Fig. 12. Efficient points for CTP8.

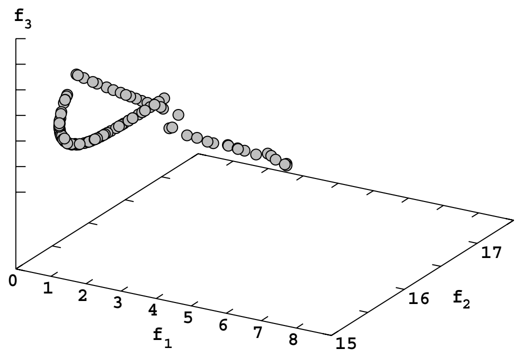

Fig. 13. Efficient points for VNT.

[33]. Total number of function evaluations specified was 20,000. The results of omni-optimizer using the same performance metrics on the ZDT problems are presented in Table 5. In each case, 99 independent simulation runs are performed and the average and standard deviation for all metrics are also reported.

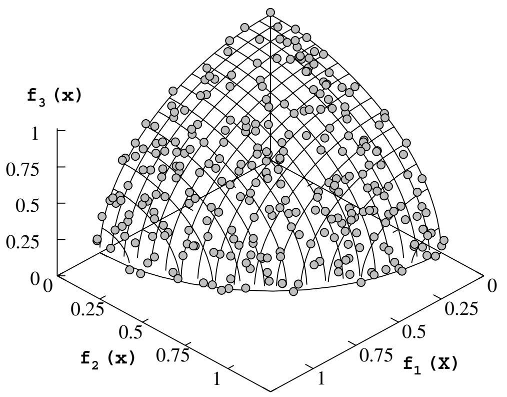

Fig. 14. Efficient points for DTLZ4.

We note that the convergence measure and hyper-volume measure for the omni-optimizer (with $\eta_c = \eta_m = 1$) are inferior to that of other algorithms. It can be attributed to the fact that small values of η_c and η_m attempts to maintain a large diversity of solution, thereby making a sacrifice in the achieved accuracy at the end. However, when $\eta_c = \eta_m = 20$ are used (results marked with 'OMNI*' in the table), the performance measures are much better and comparable to other state-of-the-art EMO methodologies. Another aspect of the use of the ϵ-domination concept with $\epsilon = 0.001$ in these problems is that the omni-optimizer prevents the algorithm from attaining a final crisp function value. Function values within ϵ are all emphasized equally well and therefore achieving a large convergence measure gets difficult with a spread-out population. Nevertheless, a small value of convergence metric and large values of sparsity and hyper-volume metrics indicate that the obtained population using the omni-optimizer are very close to the true Pareto-optimal frontier. However, when a value of $\epsilon = 0$ was used (marked as 'OMNI^0' in the table), best values of performance metrics on difficult problems are obtained. This study shows the advantage of performing a parametric study, but even without such parameter tuning reasonable performances across various problems are achieved with the omni-optimizer.

5.4. Multi-objective, multi-optima test problem

For demonstrating the performance of omni-optimizer on multi-objective, multi-optima problems, we design two functions. The first function is as follows:

$$\begin{aligned} &\text{Minimize} \quad f_1(x) = \sum_{i=1}^{n} \sin(\pi x_i), \\ &\text{Minimize} \quad f_2(x) = \sum_{i=1}^{n} \cos(\pi x_i), \\ &\qquad\qquad 0 \leqslant x_i \leqslant 6, \quad i = 1, 2, \ldots, n. \end{aligned} \tag{16}$$

Here, both objectives are periodic functions with a 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 $n = 5$ here. 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 1000 and run the algorithm for 500 generations to capture as many Pareto-optimal solutions as possible. It is interesting to note that both NSGA-II and omni-optimizer find the entire range of efficient points in the objective space, as shown in Figs. 15 and 16. However, the variable space plots show a different scenario. For this problem, obtaining the optima is not difficult but obtaining different Pareto-optimal points corresponding to every efficient point is a difficult task. The lower diagonal plots in Fig. 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 optimal solutions for different combinations of variables in all nine regions

Table 5
Performance comparison of the six MOEAs for ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6

Test problem	MOEA	Convergence measure		Sparsity		Hyper-volume	
		Avg.	SD	Avg.	SD	Avg.	SD
ZDT1	NSGA-II	0.00054898	6.62e−05	0.858	0.0202	0.8701	3.85e−04
	C-NSGA-II	0.00061173	7.86e−05	0.994	0.0043	0.8713	2.25e−04
	PESA	0.00053481	12.62e−05	0.754	0.0331	0.8680	6.76e−04
	SPEA2	0.00100589	12.06e−05	0.999	0.0014	0.8708	1.86e−04
	ϵ-MOEA	0.00039545	1.22e−05	0.991	0.0050	0.8702	1.86e−04
	OMNI	0.01089320	9.14e−04	0.922	0.0260	0.8546	9.28e−04
	OMNI*	0.00536259	4.32e−04	0.911	0.0251	0.8626	6.54e−04
	OMNI^0	0.00416721	3.80e−04	0.960	0.0151	0.8649	6.00e−04
ZDT2	NSGA-II	0.00037851	1.88e−05	0.855	0.0250	0.5372	3.01e−04
	C-NSGA-II	0.00040011	1.91e−05	0.994	0.0015	0.5374	4.42e−04
	PESA	0.00037942	2.95e−05	0.759	0.0202	0.5329	11.25e−04
	SPEA2	0.00082852	11.38e−05	0.999	0.0015	0.5374	2.61e−04
	ϵ-MOEA	0.00046448	2.47e−05	0.994	0.0037	0.5383	6.39e−05
	OMNI	0.01145880	8.19e−04	0.910	0.0244	0.5189	1.01e−03
	OMNI*	0.00522185	4.58e−04	0.898	0.0257	0.5288	8.19e−04
	OMNI^0	0.00401049	4.03e−04	0.956	0.0163	0.5312	6.39e−04
ZDT3	NSGA-II	0.00232321	13.95e−05	0.887	0.0675	1.3285	1.27e−04
	C-NSGA-II	0.00239445	12.30e−05	0.991	0.0083	1.3277	9.82e−04
	PESA	0.00211373	15.38e−05	0.882	0.0492	1.2901	7.49e−03
	SPEA2	0.00260542	15.46e−05	0.996	0.0023	1.3276	2.54e−04
	ϵ-MOEA	0.00175135	7.45e−05	0.986	0.0055	1.3287	1.31e−04
	OMNI	0.00677234	7.94e−04	0.789	0.2017	1.3009	1.91e−03
	OMNI*	0.00400560	2.65e−04	0.776	0.2644	1.3180	1.10e−03
	OMNI^0	0.00347718	2.19e−04	0.776	0.3039	1.3211	8.29e−03
ZDT4	NSGA-II	0.00639002	0.0043	0.958	0.0328	0.8613	0.00640
	C-NSGA-II	0.00618386	0.0744	0.998	0.0029	0.8558	0.00301
	PESA	0.00730242	0.0047	0.798	0.0352	0.8566	0.00710
	SPEA2	0.00769278	0.0043	0.989	0.0132	0.8609	0.00536
	ϵ-MOEA	0.00259063	0.0006	0.987	0.0076	0.8509	0.01537
	OMNI	1.69495000	1.0766	0.765	0.0708	0.4819	0.71756
	OMNI*	0.01105010	0.0097	0.899	0.0389	0.8483	0.01779
	OMNI^0	0.00545542	0.0025	0.958	0.0194	0.8566	0.01522
ZDT6	NSGA-II	0.07896111	0.0067	0.815	0.0157	0.3959	0.00894
	C-NSGA-II	0.07940667	0.0110	0.995	0.0029	0.3990	0.01154
	PESA	0.06415652	0.0073	0.748	0.0345	0.4145	0.00990
	SPEA2	0.00573584	0.0009	0.998	0.0029	0.4968	0.00117
	ϵ-MOEA	0.06792800	0.0118	0.996	0.0023	0.4112	0.01573
	OMNI	0.05342110	0.0055	0.863	0.0228	0.4176	0.00645
	OMNI*	0.02528650	0.0024	0.872	0.0288	0.4511	0.00291
	OMNI^0	0.02141800	0.0023	0.956	0.0143	0.4559	0.00275

Average is indicated by 'Avg.' and standard deviation is indicated by 'SD'. Omni-optimizer runs are as follows: 'OMNI' is for $\eta_c = \eta_m = 1$ and $\epsilon = 0.001$, 'OMNI*' is for $\eta_c = \eta_m = 20$ and $\epsilon = 0.001$, and 'OMNI^0' is for $\eta_c = \eta_m = 20$ and $\epsilon = 0$.

are obtained using the omni-optimizer. Since no variable-space crowding was considered in the original NSGA-II, not all optimal combinations are found by NSGA-II.

The second function that we have designed is as follows:

$$\begin{aligned} &\text{Minimize} \quad f_1(x) = \sin\left(\pi \sum_{i=1}^{n} x_i\right), \\ &\text{Minimize} \quad f_2(x) = \cos\left(\pi \sum_{i=1}^{n} x_i\right), \\ &\qquad 0 \leqslant x_i \leqslant 1, \quad i = 1, 2, \ldots, n. \end{aligned} \tag{17}$$

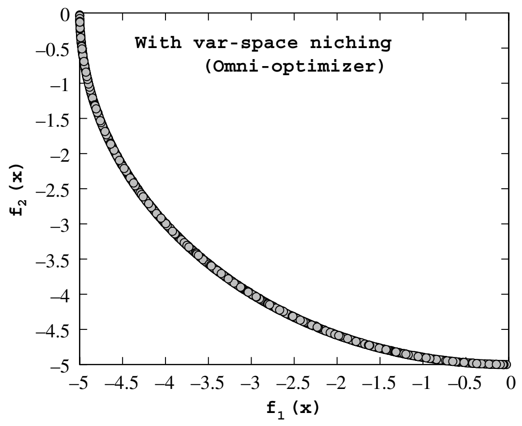

Fig. 15. Efficient points using omni-optimizer.

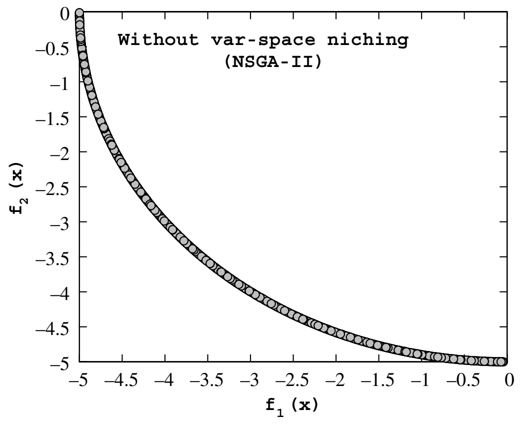

Fig. 16. Efficient points using NSGA-II.

Let $y = \sum_{i=1}^{n} x_i$, then range of y is [0, 5] and the Pareto-optimal region corresponds to $1 \leqslant y \leqslant 1.5$ and $3 \leqslant y \leqslant 3.5$. This implies that any solution that gives the value of y in the above region corresponds to the Pareto-optimal front. Since the variables in consideration are continuous variables, there exist infinite such solutions (with different $\mathbf{x}$) corresponding to every point (y) on the efficient frontier. In such a case, an ideal algorithm should find an uniformly distributed population in the entire Pareto-optimal range for each variable. We run the algorithm with a population size of *500* for *500* generations. Fig. 18 shows the distribution of y in the population. As expected, solutions exist in both ranges of y mentioned above. Interestingly, the complete Pareto-optimal front (in the objective space) can be achieved by just finding a good spread of solutions in any one of the ranges of y. Since both ranges cause Pareto-optimality, the omni-optimizer is able to find a good distribution of solutions on both ranges. On the other hand, Fig. 19 shows that the original NSGA-II is able to find only one of the ranges of y, which is enough to find a good trade-off distribution on the objective space. To show the corresponding $\mathbf{x}$ values for each algorithm, Fig. 20 plots all $\binom{5}{2}$ or 10 pair-wise variable combinations. The lower half of the figures show that the omni-optimizer is able to find a widely distributed set of Pareto-optimal solutions. The difference between NSGA-II (upper half of figures) and omni-optimizer (lower half of figures) is clear from the figure. This study demonstrates the flexibility and versatility of the proposed omni-optimizer.

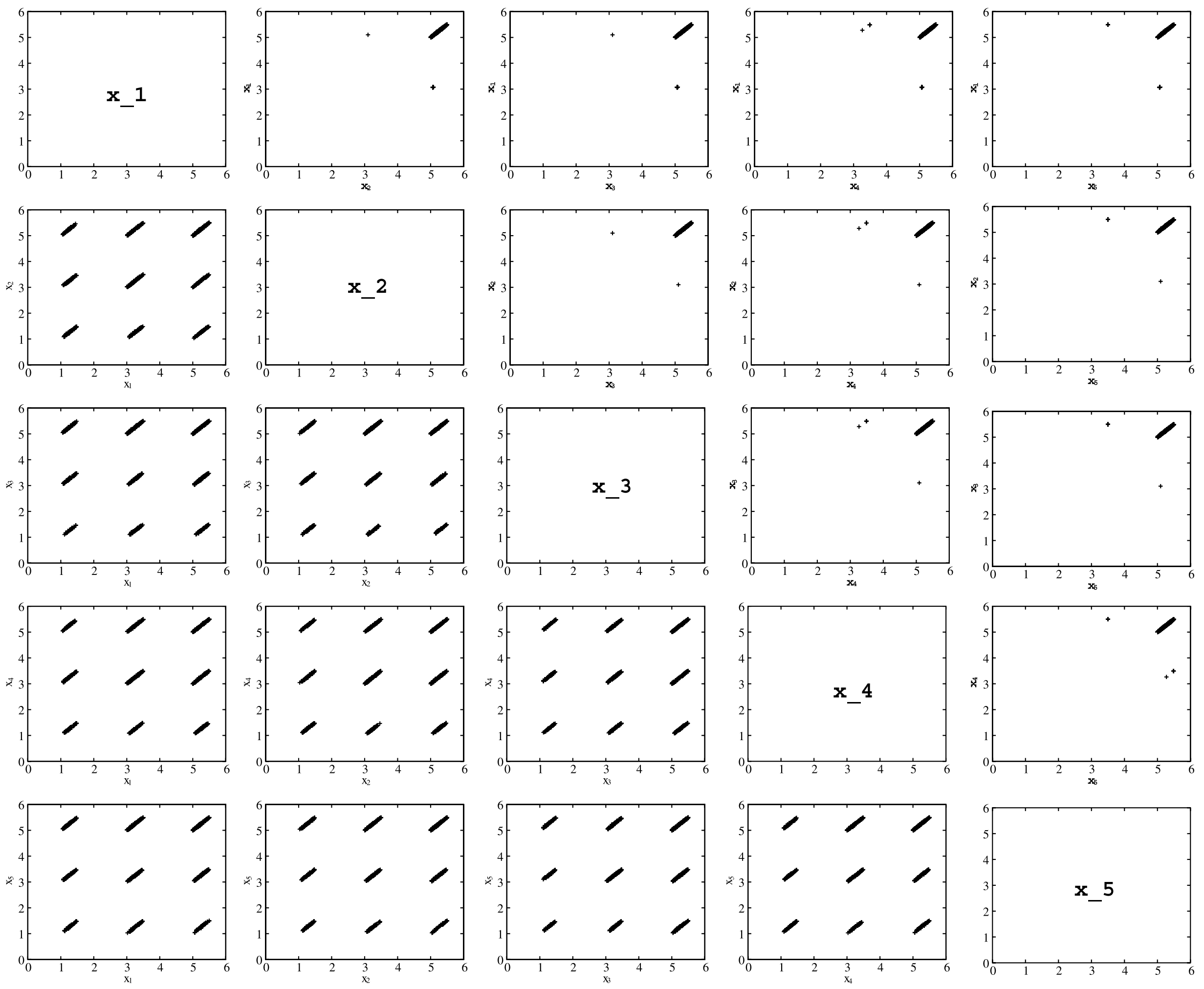

Fig. 17. Pareto-optimal solutions with omni-optimizer (left) and NSGA-II (right). The axes in a (i,j)-plot correspond to variables x_i and x_j.

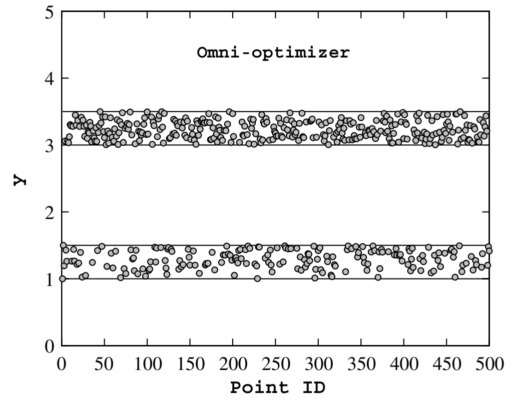

Fig. 18. Distribution of $y = \sum_{i=1}^{5} x_i$ for the omni-optimizer.

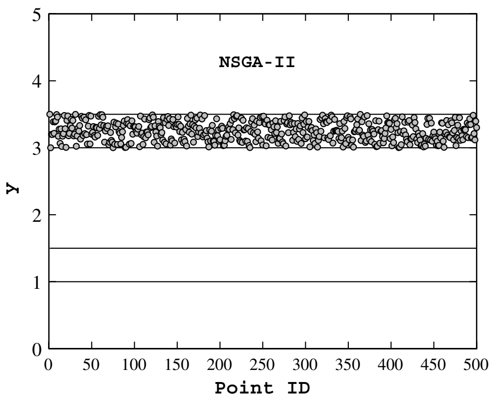

Fig. 19. Distribution of $y = \sum_{i=1}^{5} x_i$ for the NSGA-II.

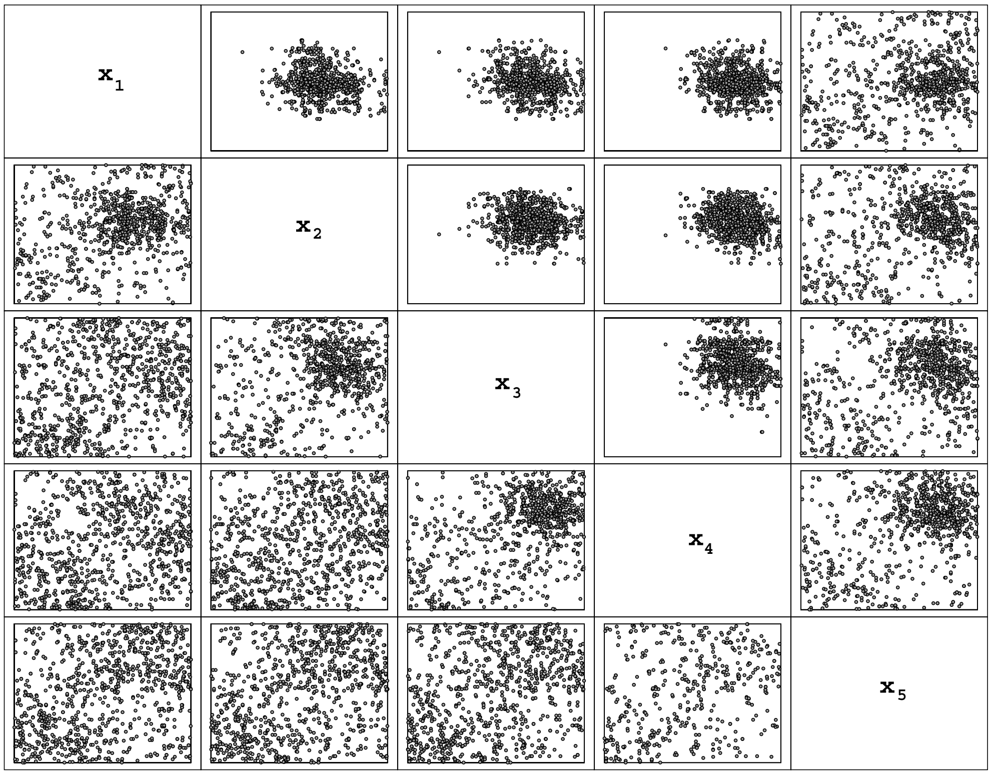

Fig. 20. Scatter plot of decision variables with omni-optimizer (left) and NSGA-II (right). The axes in a (i, j)-plot correspond to variables x_i and x_j.

6. Inferences from simulation runs and future ideas

As is evident from the simulation runs, the proposed omni-optimizer performs quite well on the test problems chosen for the study. The simulation results have been shown for four types of optimization problems. Some aspects of the algorithm which can be further improved are discussed below.

1. *Diversity estimate*: Crowding distance has been chosen as the metric for comparing the diversity of the solutions. For problems with a large dimension (with a larger number of variables), this metric is not very efficient and should be replaced by a more efficient diversity metric. K-mean clustering approach, adopted in

SPEA2 [11], is shown to provide a better distribution than the crowding distance approach, but is not directly applicable to the omni-optimizer framework because it does not assign a numerical value (as a measure of diversity) to every individual in a population rather prunes the populations by removing more crowded solutions. Further, in this algorithm, the diversity is required both in genotypic and phenotypic space, hence the chosen metric should be able to assign suitable normalized values to a solution in both the genotypic and phenotypic space. Moreover, the K-mean clustering is a computationally expensive proposition, particularly when it is to be applied in both genotypic and phenotypic spaces. Other computationally quicker yet scalable diversity measures may be thought of and applied.
2. *ϵ-domination*: In order to maintain diversity, we have used the concept of ϵ-domination, which allows all solutions within an difference of ϵ_i in the ith objective to belong in the same class during the ranking procedure. At the end, when a population contains an optimal solution, slightly inferior solutions are also given a similar emphasis along with the optimal solution. Since diversity preservation is also a goal in the omni-optimizer, this process may allow the algorithm to lose the already-obtained optimal solution at the expense of trying to keep a diverse yet non-optimal solution. One way to overcome this difficulty would be to restart the simulation when no further improvement in the best solutions is observed with the already obtained solution set as initial points and choose a progressively smaller value for ϵ. This issue is deeply related to the age-old exploration-exploitation issue [19,18] and still remains as a balancing act for a successful population-based optimization procedure.
3. *Multi-modal optimization*: The omni-optimizer as designed attempts to find only the globally optimal solutions. It does not have any mechanism that can ensure that locally optimal solutions, if desired, are also retained in the population. Achieving this is difficult especially with multi-objective problems since multiple fronts will then have to co-exist for a multi-modal, multi-objective problem.

Despite the success of one optimization strategy to solve various kinds of problems by means of degeneracy, the study is no way in violation of the no-free-lunch (NFL) theorem [34]. The algorithm has been designed to be used as a generic and robust optimizer that can handle a wide variety or problems. However, this characteristic prevents the algorithm from exploiting any special feature that a specific problem may offer (like if a problem is uni-modal, choosing a value of $\epsilon = 0$ may speed up the convergence rate, but this information about an unknown problem may not be available a priori). Specialized algorithms performing better than omni-optimizer on a specific problem still exist and research efforts must still be focussed on developing such specialized algorithms. But, the development of such a generic optimization procedure provides a holistic view of optimization algorithms in its way of connecting different types of problem-solving tasks and definitely remains as a starting optimization routine for a given problem. After a representative set of solutions have been obtained, a more specialized algorithm (for example, a classical algorithm or a multi-objective local search, as the case may be) could then be used for individuals in the population to make a reliable hybrid and robust [18] optimization procedure.

7. Conclusion

The optimization literature deals with single and multi-objective optimization problems and uni-optimal and multi-optima problems differently. In this paper, for the first time, we have presented an omni-optimizer which is designed to solve different types of optimization problems usually encountered in practice—multi-objective, multi-optima 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. We have also proposed in this paper some new concepts like restricted selection and crowding measure utilizing both objective and variable space information. A more robust and disruptive mutation scheme is proposed which makes the omni-optimizer more resilient to local optima. More research is needed to compare the proposed approach with dedicated single and multi-objective optimization procedures and on more synthetic and real-world optimization problems.

References

[1] K. Deb, 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, vol. 3410, 2005, pp. 41–65.

[2] K. Deb, Optimization for Engineering Design: Algorithms and Examples, Prentice-Hall, New Delhi, 1995.

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

[4] K. Miettinen, Nonlinear Multiobjective Optimization, Kluwer, Boston, 1999.

[5] Y. Sensor, Pareto optimality in multi-objective problems, Applied Mathematics and Optimization 4 (1) (1977) 41–59.

[6] D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best, in: Proceedings of the Third International Conference on Genetic Algorithms, 1989, pp. 116–121.

[7] L.J. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination, in: Proceedings of the Foundations of Genetic Algorithms 1 (FOGA-1), 1991, pp. 265–283.

[8] K. Deb, A. Anand, D. Joshi, A computationally efficient evolutionary algorithm for real-parameter optimization, Evolutionary Computation Journal 10 (4) (2002) 371–395.

[9] T.T. Binh, U. Korn, MOBES: A multiobjective evolution strategy for constrained optimization problems, in: Proceedings of the Third International Conference on Genetic Algorithms (Mendel 97), 1997, pp. 176–182.

[10] J.D. Knowles, D.W. Corne, Approximating the non-dominated front using the Pareto archived evolution strategy, Evolutionary Computation Journal 8 (2) (2000) 149–172.

[11] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization, in: K.C. Giannakoglou, D.T. Tsahalis, J. Périaux, K.D. Papailiou, T. Fogarty (Eds.), Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems, Athens, Greece, 2001, International Center for Numerical Methods in Engineering (Cmine), 2001, pp. 95–100.

[12] N. Srinivas, K. Deb, Multi-objective function optimization using non-dominated sorting genetic algorithms, Evolutionary Computation Journal 2 (3) (1994) 221–248.

[13] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA–II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.

[14] K. Deb, M. Mohan, S. Mishra, Towards a quick computation of well-spread pareto-optimal solutions, in: Proceedings of the Second Evolutionary Multi-Criterion Optimization (EMO-03) Conference (LNCS 2632), 2003, pp. 222–236.

[15] D.E. Goldberg, J. Richardson, Genetic algorithms with sharing for multimodal function optimization, in: Proceedings of the First International Conference on Genetic Algorithms and Their Applications, 1987, pp. 41–49.

[16] A. Pétrowski, A clearing procedure as a niching method for genetic algorithms. in: Proceedings of the IEEE Third International Conference on Evolutionary Computation (ICEC'96), 1996, pp. 798–803.

[17] F. Vavak and T.C. Fogarty. Comparison of steady state and generational genetic algorithms for use in nonstationary environments, in: Proceedings of IEEE Conference on Evolutionary Computation, Nagoya, Japan, 20–22 May 1996, pp. 192–195.

[18] D.E. Goldberg, Genetic Algorithms for Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[19] J.H. Holland, Adaptation in Natural and Artificial Systems, MIT Press, Ann Arbor, MI, 1975.

[20] K. Deb, R.B. Agrawal, Simulated binary crossover for continuous search space, Complex Systems 9 (2) (1995) 115–148.

[21] H. Kita, I. Ono, S. Kobayashi, The multi-parent unimodal normal distribution crossover for real-coded genetic algorithms, Tokyo Institute of Technology, Japan, 1998.

[22] K. Deb, Multi-objective optimization using evolutionary algorithms, Wiley, Chichester, UK, 2001.

[23] T.R. Cruse, Reliability-based Mechanical Design, Marcel Dekker, New York, 1997.

[24] K. Deb, An efficient constraint handling method for genetic algorithms, Computer Methods in Applied Mechanics and Engineering 186 (2-4) (2000) 311–338.

[25] K. Deb, M. Goyal, A combined genetic adaptive search (geneas) for engineering design, Computer Science and Informatics 26 (4) (1996) 30–45.

[26] K. Deb, M. Mohan, S. Mishra, Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions, Evolutionary Computation Journal 13 (4) (2005) 501–525.

[27] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast and elitist multi-objective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.

[28] F. Hoffmeister, J. Sprave, Problem-independent handling of constraints by use of metric penalty functions, in: L.J. Fogel, P.J. Angeline, T. Back (Eds.), Proceedings of the Fifth Annual Conference on Evolutionary Programming, The MIT Press, San Diego, California, 1996, pp. 289–294.

[29] H.-P. Schwefel, Evolution and Optimum Seeking, Wiley, New York, 1995.

[30] S.-Y. Ho, L.-S. Shu, J.-H. Chen, Intelligent evolutionary algorithms for large parameter optimization problems, IEEE Transactions on Evolutionary Computation 8 (6) (2004) 522–541.

[31] K. Deb, Genetic Algorithms in Multi-modal Function Optimization, Master's thesis, University of Alabama, Tuscaloosa, AL, 1989.

[32] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, Scalable multi-objective optimization test problems, in: Proceedings of the Congress on Evolutionary Computation (CEC-2002), 2002, pp. 825–830.
[33] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms – a comparative case study, Parallel Problem Solving from Nature V (PPSN-V) (1998) 292–301.
[34] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1) (1977) 67–82.