Table 2 Computational complexity of finding the non-dominated set for different values of M.

M	Approach 1	Approach 2	Approach 3
4	$14.714N^{1.5447}$	$25.354N^{1.1248}$	$14.120N^{1.1228}$
10	$12.614N^{1.9543}$	$8.145N^{1.8834}$	$7.431N^{1.8826}$
20	$19.525N^{2.0034}$	$9.772N^{2.0032}$	$9.413N^{2.0013}$

above time complexities reflect the average time complexity of an approach and that the orders of computational complexity presented in each approach are calculated for the worst case scenarios. Nevertheless, the experimental results show that with large values of M, the complexity approaches $O(N^2)$ in the range $100 \leq N \leq 1000$.

It is clear that the non-dominated solutions found by any of the above approaches are the *best* solutions in a population. Thus, these solutions are also called the solutions of the best non-dominated set.

2.4.7 Non-Dominated Sorting of a Population

We shall see later in Chapter 5 that most evolutionary multi-objective optimization algorithms require us to find only the best non-dominated front in a population. These algorithms classify the population into two sets – the non-dominated set and the remaining dominated set. However, there exist some algorithms which require the entire population to be classified into various non-domination levels. In such algorithms, the population needs to be sorted according to an ascending level of non-domination. The best non-dominated solutions are called non-dominated solutions of level 1. In order to find solutions for the next level of non-domination, there is a simple procedure which is usually followed. Once the best non-dominated set is identified, they are temporarily disregarded from the population. The non-dominated solutions of the remaining population are then found and are called non-dominated solutions of level 2. In order to find the non-dominated solutions of level 3, all non-dominated solutions of levels 1 and 2 are disregarded and new non-dominated solutions are found. This procedure is continued until all population members are classified into a non-dominated level. It is important to reiterate that non-dominated solutions of level 1 are better than non-dominated solutions of level 2, and so on.

Non-Dominated Sorting

Step 1 Set all non-dominated sets P_j, $(j = 1, 2, \ldots)$ as empty sets. Set non-domination level counter $j = 1$.

Step 2 Use any one of the Approaches 1 to 3 to find the non-dominated set P' of population P.

Step 3 Update $P_j = P'$ and $P = P \backslash P'$.

Step 4 If $P \neq \emptyset$, increment j by one and go to Step 2. Otherwise, stop and declare all non-dominated sets P_i, for $i = 1, 2, \ldots, j$.

We illustrate the working of the above procedure on the same set of five solutions (Figure 14) by using Approach 1 for identifying the non-dominated set.

Step 1 We first set $j = 1$ to identify the non-dominated solutions of the first level.

Step 2 As shown earlier, the first non-dominated set is $P' = \{3, 5\}$.

Step 3 Update $P = P' = \{3, 5\}$ and modify P by deleting solutions 3 and 5 from it, or $P = \{1, 2, 4\}$.

Step 4 Since P is a non-empty set, we move to Step 2 in search of solutions of the second non-domination level ($j = 2$).

Step 2 We now move to Approach 1, outlined above on page 34.

- **Step 1** Set $i = 1$ and $P' = \emptyset$.
- **Step 2** Solution 2 does not dominate solution 1.
- **Step 3** We now check solution 4 with solution 1.
- **Step 2** Solution 4 does not dominate solution 1 either.
- **Step 3** We set $P' = \{1\}$.
- **Step 4** We increment i to 2 (check for solution 2) and move to Step 2.
- **Step 2** Solution 1 dominates solution 2.
- **Step 3** Thus, we move to Step 4.
- **Step 4** We now check solution 4 for its inclusion in the second non-dominated set.
- **Step 2** Solution 1 does not dominate solution 4.
- **Step 3** Check with solution 2.
- **Step 2** Solution 2 does not dominate solution 4 either.
- **Step 3** We include solution 4 in the set or $P' = \{1, 4\}$.
- **Step 4** All three solutions are checked for non-domination. Thus, we declare the non-dominated set as $P' = \{1, 4\}$.

Step 3 Update $P_2 = P' = \{1, 4\}$ and modify P by deleting P' from it, or $P = \{2\}$.

Step 4 We increment j to 3 and move to Step 2 in search of solutions in the third non-dominated set.

Since there is only one solution left in P, this must belong to the third non-dominated set. Thus, there are three non-dominated sets in the population with the following

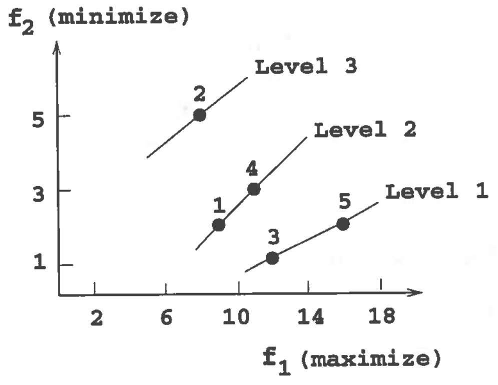

Figure 18 Solutions classified into various non-domination classes.

classification in a decreasing order of non-domination (or an increasing order of domination):

$$((3,5),(1,4),(2))$$

The corresponding sets are marked in Figure 18 with hypothetical curves. The other two approaches can also be used and an identical outcome will be found.

The complexity of this complete classification procedure is the sum of the individual complexities involved in identifying each non-dominated set. It is important to note that after the first non-dominated set is identified, the number of solutions left in P is smaller than the original population size. Thus, subsequent identifications will require smaller computational complexity. In an average scenario, the number of non-domination levels is usually much smaller than the population size. We shall show later in Section 8.8 that as M increases the number of different non-domination levels in a random population reduces drastically. Thus, for all practical purposes, the computational complexity of the overall non-dominated sorting procedure is governed by the procedure of identifying the first non-dominated set.

In a similar experimental study, we find the overall computational complexity needed in sorting several populations according to different non-domination levels. The rate of increase in the complexity is slightly more compared to just finding the best non-dominated set. Interestingly, even in the complete non-dominated sorting problem, Approach 3 turns out to be better than the other two approaches. Table 3 shows the estimated complexity measures (from simulation results) for different population sizes (N) and number of objectives (M). It is clear from these computations that Approach 3 requires the least amount of computational effort to sort a population into different non-domination levels.

An $O(MN^2)$ Non-Dominated Sorting Procedure

Using the above procedure, each front can be identified with at most $O(MN^2)$ computations. In certain scenarios, this procedure may demand more than $O(MN^2)$

Table 3 Computational complexities for a complete non-dominated sorting procedure.

M	Approach 1	Approach 2	Approach 3
4	$3.876N^{1.9722}$	$3.274N^{1.7463}$	$3.145N^{1.7135}$
10	$10.942N^{1.9722}$	$6.786N^{1.9194}$	$6.369N^{1.9133}$
20	$19.526N^{2.0034}$	$9.768N^{2.0033}$	$9.408N^{2.0014}$

computational effort for the overall non-dominated sorting of a population. Here, we suggest a completely different procedure which uses a better bookkeeping strategy requiring $O(MN^2)$ overall computational complexity.

First, for each solution we calculate two entities: (i) *domination count* n_i, the number of solutions which dominate the solution i, and (ii) S_i, a set of solutions which the solution i dominates. This requires $O(MN^2)$ comparisons. At the end of this procedure, all solutions in the first non-dominated front will have their domination count as zero. Now, for each of these solutions (each solution i with $n_i = 0$), we visit each member (j) of its set S_i and reduce its domination count by one. In doing so, if for any member j the domination count becomes zero, we put it in a separate list P'. After such modifications on S_i are performed for each i with $n_i = 0$, all solutions of P' would belong to the second non-dominated front. The above procedure can be continued with each member of P' and the third non-dominated front can be identified. This process continues until all solutions are classified.

An $O(MN^2)$ Non-Dominated Sorting Algorithm

Step 1 For each $i \in P$, $n_i = 0$ and initialize $S_i = \emptyset$. For all $j \neq i$ and $j \in P$, perform Step 2 and then proceed to Step 3.

Step 2 If $i \preceq j$, update $S_p = S_p \cup \{j\}$. Otherwise, if $j \preceq i$, set $n_i = n_i + 1$.

Step 3 If $n_i = 0$, keep i in the first non-dominated front P_1 (we called this set P' in the above paragraph). Set a front counter $k = 1$.

Step 4 While $P_k \neq \emptyset$, perform the following steps.

Step 5 Initialize $Q = \emptyset$ for storing next non-dominated solutions. For each $i \in P_k$ and for each $j \in S_i$,

Step 5a Update $n_j = n_j - 1$.

Step 5b If $n_j = 0$, keep j in Q, or perform $Q = Q \cup \{j\}$.

Step 6 Set $k = k + 1$ and $P_k = Q$. Go to Step 4.

Steps 1 to 3 find the solutions in the first non-dominated front and require $O(MN^2)$ computational complexity. Steps 4 to 6 repeatedly find higher fronts and require at most $O(N^2)$ comparisons, as argued below.

For each solution i in the second or higher-level of non-domination, the domination count n_i can be at most $N-1$. Thus, each solution i will be visited at most $N-1$ times before its domination count becomes zero. At this point, the solution is assigned a particular non-domination level and will never be visited again. Since there are at most $N-1$ such solutions, the complexity of identifying second and more fronts is $O(N^2)$. Thus, the overall complexity of the procedure is $O(MN^2)$. It is important to note that although the time complexity has reduced to $O(MN^2)$, the storage requirement has increased to $O(N^2)$.

2.5 Optimality Conditions

In this section, we will outline the theoretical Pareto-optimality conditions for a constrained MOOP. Here, we state the optimality conditions for the MOOP given in equation (2.1). We assume that all objectives and constraint functions are continuously differentiable. As in single-objective optimization, there exist first- and second-order optimality conditions for multi-objective optimization. Here, we state the first-order necessary condition and a specific sufficient condition. Interested readers may refer to more advanced classical books on multi-objective optimization (Chankong and Haimes, 1983; Ehrgott, 2000; Miettinen, 1999) for a more comprehensive treatment of different optimality conditions.

The following condition is known as the necessary condition for Pareto-optimality.

Theorem 2.5.1. *(Fritz–John necessary condition). A necessary condition for* $\mathbf{x}^*$ *to be Pareto-optimal is that there exist vectors* $\boldsymbol{\lambda} \geq \mathbf{0}$ *and* $\mathbf{u} \geq \mathbf{0}$ *(where* $\boldsymbol{\lambda} \in \mathbb{R}^M$, $\mathbf{u} \in \mathbb{R}^J$ *and* $\boldsymbol{\lambda}, \mathbf{u} \neq \mathbf{0}$*) such that the following conditions are true:*

1. $\sum_{m=1}^{M} \lambda_m \nabla f_m(\mathbf{x}^*) - \sum_{j=1}^{J} u_j \nabla g_j(\mathbf{x}^*) = \mathbf{0}$, *and*
2. $u_j g_j(\mathbf{x}^*) = 0$ *for all* $j = 1, 2, \ldots, J$.

For a proof, readers may refer to Cunha and Polak (1967). Miettinen (1999) argues that the above theorem is also valid as the necessary condition for a solution to be weakly Pareto-optimal. Those readers familiar with the Kuhn–Tucker necessary conditions for single-objective optimization will immediately recognize the similarity between the above conditions and that of the single-objective optimization. The difference is in the inclusion of a λ-vector with the gradient vector of the objectives.

For an unconstrained MOOP, the above theorem requires the following condition:

$$\sum_{m=1}^{M} \lambda_m \nabla f_m(\mathbf{x}^*) = \mathbf{0}$$

to be necessary for a solution to be Pareto-optimal. Writing the above vector equation in matrix form, we have the following necessary condition for an M-objective and n-variable unconstrained MOOP:

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_1} \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_M}{\partial x_2} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_1}{\partial x_n} & \frac{\partial f_2}{\partial x_n} & \cdots & \frac{\partial f_M}{\partial x_n} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_M \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.7}$$

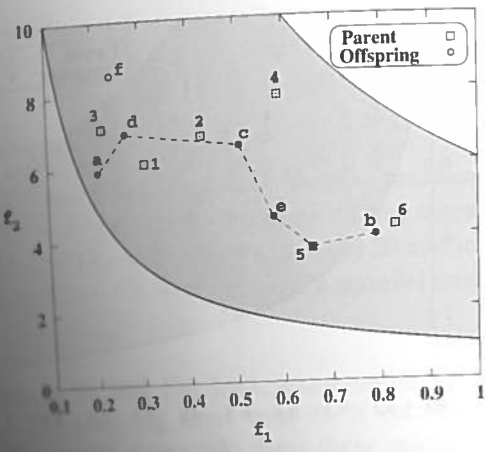

Figure 136 The parent population of the next generation consists of solutions joined by dashed lines.

6.1.3 Computational Complexity

In Step 1, the best non-dominated solutions are found. This requires O(Mλ^2) comparisons. In Step 2, for each non-dominated solution in the offspring population, [illegible] in the parent population are checked, thereby requiring the worst case [illegible] comparisons. If $\mu = \lambda = N$ is used, the worst case complexity is O(MN^2). However, it is important to remember that this algorithm does not attempt to maintain [illegible] diversity in the obtained solutions.

[illegible] Advantages

[illegible] has proven that the above algorithm with a 'positive variation kernel' [illegible] convergence (of at least one solution in the population) [illegible] in a finite number of trials (refer to Section 8.9 below) [illegible]. The positiveness of the variation kernel (representing [illegible] of crossover and mutation operators) makes sure that [illegible] from an arbitrary set of parent solutions in a [illegible]. Thus, in any population, if no Pareto-optimal solution [illegible] of the combined search operators ensures [illegible] in a finite number of trials. With the elite- [illegible] this member cannot ever be deleted from [illegible] will not be dominated by any other [illegible] will always lie in P_t.

[illegible] of diversity among [illegible] algorithm converges

to a single Pareto-optimal solution, it cannot be justified for its use over other classical algorithms. What makes evolutionary algorithms attractive is their ability to find multiple Pareto-optimal solutions in one single run.

Whenever all μ parent solutions are Pareto-optimal, no new Pareto-optimal solutions can be accommodated in the new population. This shortcoming does not allow the above algorithm to find a more diverse set of Pareto-optimal solutions. Although a niche formation scheme can be used in choosing the elements from Q', Q^* and Q_t to fill up the population with a diverse set of solutions from these sets, the algorithm stagnates as soon as an arbitrary set of μ Pareto-optimal solutions are found as parents.

The set P' is given priority over Q' in forming the new population P_{t+1}, although all members of Q' are non-dominated with all members of P_t. Since a member of P_t can dominate some members of P', the resulting set P_{t+1} may not contain all non-dominated solutions.

Since this algorithm does not ensure any diversity among the obtained solutions, we have not used it for testing on the example problem.

6.2 Elitist Non-Dominated Sorting Genetic Algorithm

This author and his students suggested an elitist non-dominated sorting GA (termed NSGA-II) in 2000 (Deb et al., 2000a, 2000b). Unlike the above method of using only an elite-preservation strategy as in the previous algorithm, NSGA-II also uses an explicit diversity-preserving mechanism. In most aspects, this algorithm does not have much similarity with the original NSGA, but the authors kept the name NSGA-II to highlight its genesis and place of origin.

Like Rudolph's algorithm, in NSGA-II, the offspring population Q_t is first created by using the parent population P_t. However, instead of finding the non-dominated front of Q_t only, first the two populations are combined together to form R_t of size 2N. Then, a non-dominated sorting is used to classify the entire population R_t. Although this requires more effort compared to performing a non-dominated sorting on Q_t alone, it allows a global non-domination check among the offspring and parent solutions. Once the non-dominated sorting is over, the new population is filled by solutions of different non-dominated fronts, one at a time. The filling starts with the best non-dominated front and continues with solutions of the second non-dominated front, followed by the third non-dominated front, and so on. Since the overall population size of R_t is 2N, not all fronts may be accommodated in N slots available in the new population. All fronts which could not be accommodated are simply deleted. When the last allowed front is being considered, there may exist more solutions in the last front than the remaining slots in the new population. This scenario is illustrated in Figure 137. Instead of arbitrarily discarding some members from the last front, it would be wise to use a niching strategy to choose the members of the last front, which reside in the least crowded region in that front.

A strategy like the above does not affect the proceedings of the algorithm much

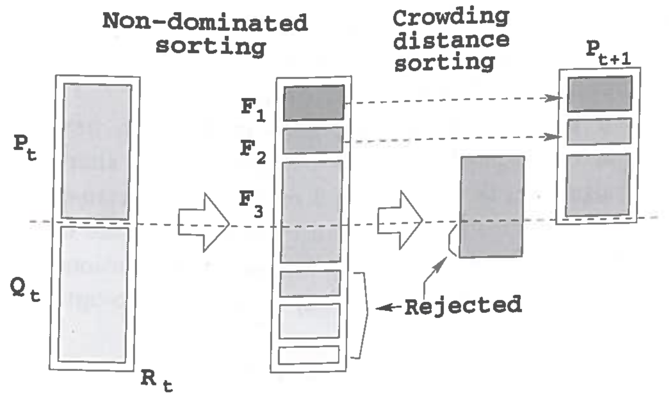

Figure 137 Schematic of the NSGA-II procedure.

in the early stages of evolution. This is because, early on, there exist many fronts in the combined population. It is likely that solutions of many good non-dominated fronts are already included in the new population, before they add up to N. It then hardly matters which solution is included to fill up the population. However, during the latter stages of the simulation, it is likely that most solutions in the population lie in the best non-dominated front. It is also likely that in the combined population R_t of size 2N, the number of solutions in the first non-dominated front exceeds N. The above algorithm then ensures that niching will choose a diverse set of solutions from this set. When the entire population converges to the Pareto-optimal front, the continuation of this algorithm will ensure a better spread among the solutions.

In the following, we outline the algorithm in a step-by-step format. Initially, a random population P_0 is created. The population is sorted into different non-domination levels. Each solution is assigned a fitness equal to its non-domination level (1 is the best level). Thus, minimization of the fitness is assumed. Binary tournament selection (with a crowded tournament operator described later), recombination and mutation operators are used to create a offspring population Q_0 of size N. The NSGA-II procedure is outlined in the following.

NSGA-II

Step 1 Combine parent and offspring populations and create $R_t = P_t \cup Q_t$. Perform a non-dominated sorting to R_t and identify different fronts: $\mathcal{F}_i$, $i = 1, 2, \ldots$, etc.

Step 2 Set new population $P_{t+1} = \emptyset$. Set a counter $i = 1$. Until $|P_{t+1}| + |\mathcal{F}_i| < N$, perform $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$ and $i = i + 1$.

Step 3 Perform the `Crowding-sort`$(\mathcal{F}_i, <_c)$ procedure (described below on page 236) and include the most widely spread $(N - |P_{t+1}|)$ solutions by using the crowding distance values in the sorted $\mathcal{F}_i$ to P_{t+1}.

Step 4 Create offspring population Q_{t+1} from P_{t+1} by using the crowded tournament selection, crossover and mutation operators.

In Step 3, the crowding-sorting of the solutions of front i (the last front which could not be accommodated fully) is performed by using a *crowding distance metric*, which we will describe a little later. The population is arranged in descending order of magnitude of the crowding distance values. In Step 4, a crowding tournament selection operator, which also uses the crowding distance, is used.

It is important to mention here that the non-dominated sorting in Step 1 and filling up population P_{t+1} can be performed together. In this way, every time a non-dominated front is found, its size can be used to check if it can be included in the new population. If this is not possible, no more sorting is needed. This will help in reducing the run-time of the algorithm.

6.2.1 Crowded Tournament Selection Operator

The crowded comparison operator ($<_c$) compares two solutions and returns the winner of the tournament. It assumes that every solution i has two attributes:

1. A non-domination rank r_i in the population.
2. A local crowding distance (d_i) (which we define in the next subsection) in the population.

The crowding distance d_i of a solution i is a measure of the search space around i which is not occupied by any other solution in the population. Based on these two attributes, we can define the crowded tournament selection operator as follows.

Definition 6.1. *Crowded Tournament Selection Operator: A solution i wins a tournament with another solution j if any of the following conditions are true:*

1. *If solution i has a better rank, that is, $r_i < r_j$.*
2. *If they have the same rank but solution i has a better crowding distance than solution j, that is, $r_i = r_j$ and $d_i > d_j$.*

The first condition makes sure that chosen solution lies on a better non-dominated front. The second condition resolves the tie of both solutions being on the same non-dominated front by deciding on their crowded distance. The one residing in a less crowded area (with a larger crowding distance d_i) wins. The crowding distance d_i can be computed in various ways. Of these, the niche count metric (nc_i) and the head count metric (hc_i) are common (refer to Section 8.2 later). Both of these metrics require $O(MN^2)$ complexity. Although they can be used, they have to be used in a reverse sense. A solution having a smaller niche count or head count means a lesser dense solution and has to be preferred. Thus, in the second condition, one should choose solution i if $r_i = r_j$ and either $nc_i < nc_j$ or, in terms of head counts, $hc_i < hc_j$. However, in NSGA-II, we use a crowding distance metric, which requires $O(MN \log N)$ computations.

Crowding Distance

To get an estimate of the density of solutions surrounding a particular solution i in the population, we take the average distance of two solutions on either side of solution i along each of the objectives. This quantity d_i serves as an estimate of the perimeter of the cuboid formed by using the nearest neighbors as the vertices (we call this the *crowding distance*). In Figure 138, the crowding distance of the i-th solution in its front (marked with solid circles) is the average side-length of the cuboid (shown by a dashed box). The following algorithm is used to calculate the crowding distance of each point in the set $\mathcal{F}$.

Crowding Distance Assignment Procedure: Crowding-sort($\mathcal{F}, <_c$)

Step C1 Call the number of solutions in $\mathcal{F}$ as $l = |\mathcal{F}|$. For each i in the set, first assign $d_i = 0$.

Step C2 For each objective function $m = 1, 2, \ldots, M$, sort the set in worse order of f_m or, find the sorted indices vector: $I^m = \text{sort}(f_m, >)$.

Step C3 For $m = 1, 2, \ldots, M$, assign a large distance to the boundary solutions, or $d_{I_1^m} = d_{I_l^m} = \infty$, and for all other solutions $j = 2$ to $(l-1)$, assign:

$$d_{I_j^m} = d_{I_j^m} + \frac{f_m^{(I_{j+1}^m)} - f_m^{(I_{j-1}^m)}}{f_m^{\max} - f_m^{\min}}.$$

The index I_j denotes the solution index of the j-th member in the sorted list. Thus, for any objective, I_1 and I_l denote the lowest and highest objective function values, respectively. The second term on the right side of the last equation is the difference in objective function values between two neighboring solutions on either side of solution I_j. Thus, this metric denotes half of the perimeter of the enclosing cuboid with the nearest neighboring solutions placed on the vertices of the cuboid (Figure 138). It is interesting to note that for any solution i the same two solutions $(i+1)$ and $(i-1)$ need not be neighbors in all objectives, particularly for $M \geq 3$. The parameters $f_m^{\max}$ and $f_m^{\min}$ can be set as the population-maximum and population-minimum values of the m-th objective function.

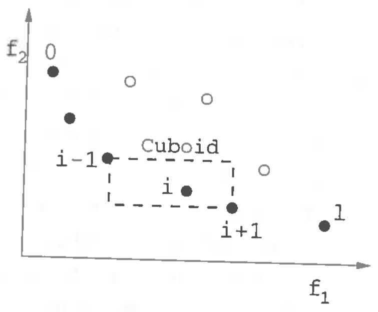

Figure 138 The crowding distance calculation. This is a reprint of Figure 1 from Deb et al. (2000b) (© Springer-Verlag Berlin Heidelberg 2000).

The above metric requires M sorting calculations in Step C2, each requiring $O(N \log N)$ computations. Step C3 requires N computations. Thus, the complexity of the above distance metric computation is $O(MN \log N)$. For large N, this complexity is smaller than $O(MN^2)$, which denotes the computational complexity required in other niching methods. Now, we illustrate the procedure while hand-simulating NSGA-II to the example problem Min-Ex.

6.2.2 Hand Calculations

We use the same parent and offspring populations as used in the previous section (Table 20).

Step 1 We first combine the populations P_t and Q_t and form $R_t = \{1, 2, 3, 4, 5, 6, a, b, c, d, e, f\}$. Next, we perform a non-dominated sorting on R_t. We obtain the following non-dominated fronts:

$$\begin{aligned}
\mathcal{F}_1 &= \{5, a, e\},\\
\mathcal{F}_2 &= \{1, 3, b, d\},\\
\mathcal{F}_3 &= \{2, 6, c, f\},\\
\mathcal{F}_4 &= \{4\}.
\end{aligned}$$

These fronts are shown in Figure 139.

Step 2 We set $P_{t+1} = \emptyset$ and $i = 1$. Next, we observe that $|P_{t+1}| + |\mathcal{F}_1| = 0 + 3 = 3$. Since this is less than the population size N $(= 6)$, we include this front in P_{t+1}. We set $P_{t+1} = \{5, a, e\}$. With these three solutions, we now need three more solutions to fill up the new parent population.

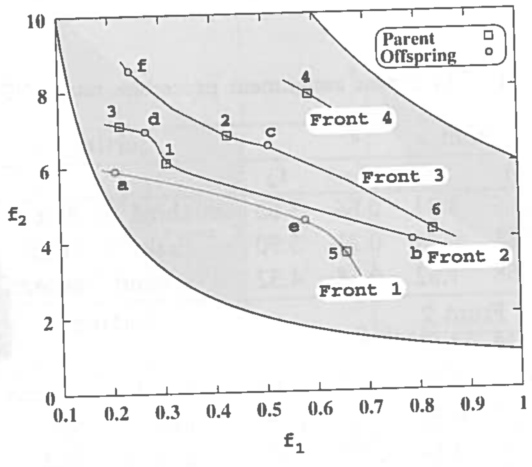

Figure 139 Four non-dominated fronts of the combined population R_t.

Now, with the inclusion of the second front, the size of $|P_{t+1}| + |\mathcal{F}_2|$ is $(3+4)$ or 7. Since this is greater than 6, we stop including any more fronts into the population. Note that if fronts 3 and 4 had not been classified earlier, we could have saved these computations.

Step 3 Next, we consider solutions of the second front only and observe that three (of the four) solutions must be chosen to fill up three remaining slots in the new population. This requires that we first sort this subpopulation (solutions 1, 3, b, and d) by using the $<_c$ operator (defined above in Section 6.2.1). We calculate the crowded distance values of these solutions in the front by using the step-by-step procedure.

Step C1 We notice that $l = 4$ and set $d_1 = d_3 = d_b = d_d = 0$. We also set $f_1^{\max} = 1$, $f_1^{\min} = 0.1$, $f_2^{\max} = 60$ and $f_2^{\min} = 0$.

Step C2 For the first objective function, the sorting of these solutions is shown in Table 21 and is as follows: $I^1 = \{3, d, 1, b\}$.

Step C3 Since solutions 3 and b are boundary solutions for f_1, we set $d_3 = d_b = \infty$. For the other two solutions, we obtain:

$$d_d = 0 + \frac{f_1^{(1)} - f_1^{(3)}}{f_1^{\max} - f_1^{\min}} = 0 + \frac{0.31 - 0.22}{1 - 0.1} = 0.10.$$

$$d_1 = 0 + \frac{f_1^{(b)} - f_1^{(d)}}{f_1^{\max} - f_1^{\min}} = 0 + \frac{0.79 - 0.27}{1 - 0.1} = 0.58.$$

Now, we turn to the second objective function and update the above distances. First, the sorting on this objective yields $I^2 = \{b, 1, d, 3\}$. Thus,

Table 21 The fitness assignment procedure under NSGA-II.

Front 1					Sorting in		Distance
Solution	x_1	x_2	f_1	f_2	f_1	f_2	
5	0.66	1.41	0.66	3.65	third	first	∞
a	0.21	0.24	0.21	5.90	first	third	∞
e	0.58	1.62	0.58	4.52	second	second	0.54
Front 2					**Sorting in**		**Distance**
Solution	x_1	x_2	f_1	f_2	f_1	f_2	
1	0.31	0.89	0.31	6.10	third	second	0.63
3	0.22	0.56	0.22	7.09	first	fourth	∞
b	0.79	2.14	0.79	3.97	fourth	first	∞
d	0.27	0.87	0.27	6.93	second	third	0.12

[illegible]

$d_b = d_3 = \infty$ and the other two distances are as follows:

$$d_1 = d_1 + \frac{f_2^{(d)} - f_2^{(b)}}{f_2^{\max} - f_2^{\min}} = 0.58 + \frac{6.93 - 3.97}{60 - 0} = 0.63.$$

$$d_d = d_d + \frac{f_2^{(3)} - f_2^{(1)}}{f_2^{\max} - f_2^{\min}} = 0.10 + \frac{7.09 - 6.10}{60 - 0} = 0.12.$$

The overall crowded distances of the four solutions are:

$$d_1 = 0.63, \quad d_3 = \infty, \quad d_b = \infty, \quad d_d = 0.12.$$

The cuboids (rectangles here) for these solutions are schematically shown in Figure 140. Solution d has the smallest perimeter of the hypercube around it than any other solution in the set $\mathcal{F}_2$, as evident from the figure. Now, we move to the main algorithm.

Step 3 A sorting according to the descending order of these crowding distance values yields the sorted set $\{3, b, 1, d\}$. We choose the first three solutions.

Step 4 The new population is $P_{t+1} = \{5, a, e, 3, b, 1\}$. These population members are shown in Figure 141 by joining them with dashed lines. It is important to note that this population is formed by choosing solutions from the better non-dominated fronts.

The offspring population Q_{t+1} has to be created next by using this parent population. We realize that the exact offspring population will depend on the chosen pair of solutions participating in a tournament and the chosen crossover and mutation operators. Let us say that we pair solutions $(5, e)$, $(a, 3)$, $(1, b)$, $(a, 1)$, (e, b) and $(3, 5)$, so that each solution participates in exactly two

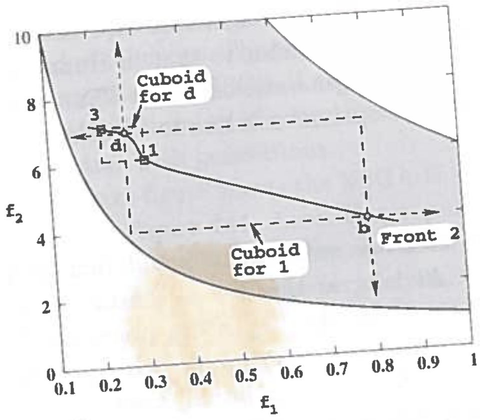

Figure 140 The cuboids of solutions 1 and d. The cuboids for solutions 3 and b extend to infinity.

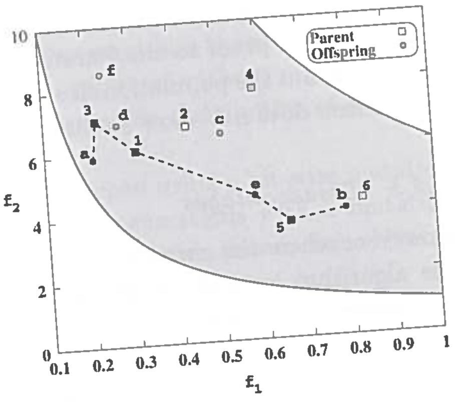

Figure 141 The new parent population P_{t+1} joined by dashed lines.

tournaments. The crowding distance values for solutions of the first front are also listed in Table 21. In the first tournament, we observe that solutions 5 and e belong to the same front ($r_5 = r_e = 1$). Thus, we choose the one with the larger crowding distance value. We find that solution 5 is the winner.

In the next tournament between solutions a and 3, solution a wins, since it belongs to a better front. Performing other tournaments, we obtain the mating pool: $\{5, a, a, b, b, e\}$. Now, these solutions can be mated pair-wise and mutated to create Q_{t+1}. This completes one generation of the NSGA-II.

6.2.3 *Computational Complexity*

Step 1 of the NSGA-II requires a non-dominated sorting of a population of size $2N$. This requires at most $O(MN^2)$ computations. Sorting in Step 3 requires crowding distance computation of all members of front i. However, the crowded tournament selection in Step 4 requires the crowding distance computation of the complete population P_{t+1} of size N. This requires $O(MN \log N)$ computations. Thus, the overall complexity of the NSGA-II is at most $O(MN^2)$.

6.2.4 *Advantages*

The diversity among non-dominated solutions is introduced by using the crowding comparison procedure which is used with the tournament selection and during the population reduction phase. Since solutions compete with their crowding distances (a measure of the density of solutions in the neighborhood), no extra niching parameter (such as σ_{share} needed in the MOGAs, NSGAs or NPGAs) is required here. Although the crowding distance is calculated in the objective function space, it can also be implemented in the parameter space, if so desired (Deb, 1999c).

In the absence of the crowded comparison operator, this algorithm also exhibits a convergence proof to the Pareto-optimal solution set similar to that in Rudolph's algorithm, but the population size would grow with the generation counter. The elitism mechanism does not allow an already found Pareto-optimal solution to be deleted.

6.2.5 *Disadvantages*

However, when the crowded comparison is used to restrict the population size, the algorithm loses its convergence property. As long as the size of the first non-dominated set is not larger than the population size, this algorithm preserves all of them. However, in latter generations, when more than N members belong to the first non-dominated set in the combined parent–offspring population, some closely-packed Pareto-optimal solutions may give their places to other non-dominated yet non-Pareto-optimal solutions. Although these latter solutions may get dominated by other Pareto-optimal solutions in a later generation, the algorithm can resort into this cycle of generating Pareto-optimal and non-Pareto-optimal solutions before finally

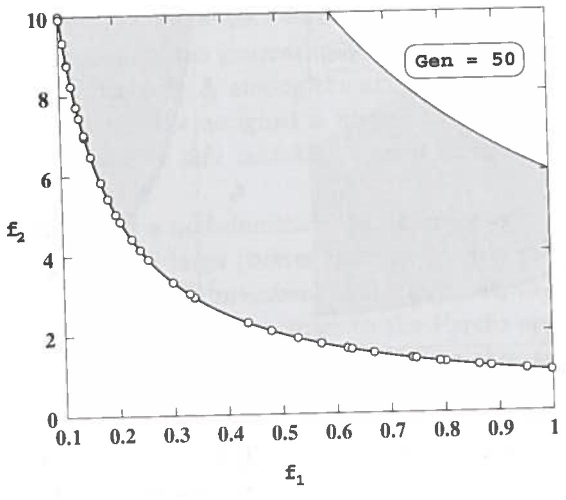

Figure 142 The population after 50 generations with the NSGA-II without mutation.

converging to a well-distributed set of Pareto-optimal solutions.

As mentioned earlier, the non-dominated sorting needs to be performed on a population of size 2N, instead of a population of size N required in most algorithms.

6.2.6 Simulation Results

We apply the above algorithm to the Min-Ex problem. The GA parameter settings are also identical to that used before. A population size of 40 and a crossover probability of 0.9 are used. The decision variables 1 and 2 are coded in 24 and 26 bits, respectively. Figure 142 shows the population of 40 solutions at the end of 50 generations without a mutation operator. This figure shows that even without the mutation operator, the NSGA-II can make the population reach the true Pareto-optimal front with a good spread of solutions. The population at the end of 500 generations, shown in Figure 143, confirms that the NSGA-II can sustain the spread of solutions without mutation even after a large number of generations are elapsed. However, the distribution of solutions gets better with generations.

The next figure shows the NSGA-II population obtained using a bit-wise mutation operator. Figure 144 shows the population after 500 generations with a mutation probability of 0.02. With the mutation operator, the exploration power of the NSGA-II gets enhanced and a better distribution of solutions is observed.

6.3 Distance-Based Pareto Genetic Algorithm

Osyczka and Kundu (1995) suggested an elitist GA, a distance-based Pareto genetic algorithm (DPGA), which attempts to emphasize the progress towards the Pareto-optimal front and the diversity along the obtained front by using one fitness measure. This algorithm maintains two populations: one standard GA population P_t where

Zitzler–Deb–Thiele's (ZDT) Test Problems

Zitzler et al. (2000) framed six problems (ZDT1 to ZDT6) based on the above construction process. After this study was reported, these test problems have further been studied by other researchers. These problems have two objectives which are to be minimized:

$$\begin{aligned} &\text{Minimize} \quad f_1(x), \\ &\text{Minimize} \quad f_2(x) = g(x)h(f_1(x), g(x)). \end{aligned} \tag{8.32}$$

The six test problems vary in the way that the three functions $f_1(x)$, $g(x)$ and $h(x)$ are defined. In all problems except ZDT5, the Pareto-optimal front is formed with $g(x) = 1$. Although these problems used f_1 as a single-variable function, the problem difficulty can be increased by using a multi-variate f_1 function or a mapped variable strategy (equation (8.30)).

ZDT1

This is a 30-variable ($n = 30$) problem having a convex Pareto-optimal set. The functions used are as follows:

$$\text{ZDT1:} \begin{cases} f_1(x) &= x_1, \\ g(x) &= 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i, \\ h(f_1, g) &= 1 - \sqrt{f_1/g}. \end{cases} \tag{8.33}$$

All variables lie in the range $[0, 1]$. The Pareto-optimal region corresponds to $0 \leq x_1^* \leq 1$ and $x_i^* = 0$ for $i = 2, 3, \ldots, 30$. Figure 213 shows a search region (which has a uniform density of solutions) and the Pareto-optimal region in the objective space. This is probably the easiest of all of the six problems, having a continuous Pareto-optimal front and a uniform distribution of solutions across the front. The

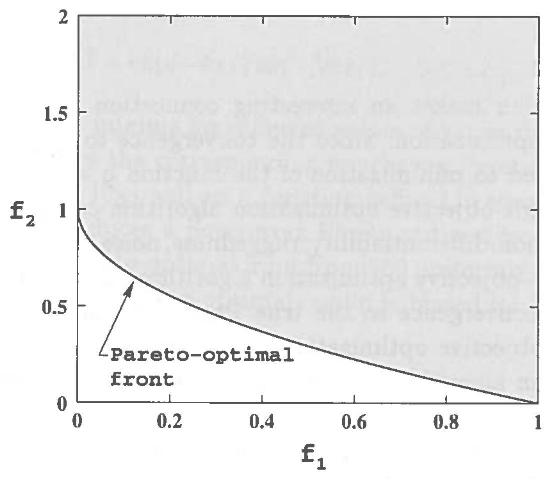

Figure 213 The search space near the Pareto-optimal region for ZDT1.

only difficulty an MOEA may face from this problem is in tackling a large number of decision variables.

ZDT2

This is also an $n = 30$ variable problem having a nonconvex Pareto-optimal set:

$$\text{ZDT2:} \begin{cases} f_1(\mathbf{x}) &= x_1, \\ g(\mathbf{x}) &= 1 + \frac{9}{n-1} \sum_{i=2}^{n} x_i, \\ h(f_1, g) &= 1 - (f_1/g)^2. \end{cases} \tag{8.34}$$

All variables lie in the range $[0, 1]$. The Pareto-optimal region corresponds to $0 \leq x_1^* \leq 1$ and $x_i^* = 0$ for $i = 2, 3, \ldots, 30$. Figure 214 shows the search region (which has a uniform density of solutions) and the Pareto-optimal region on the objective function space. The only difficulty with this problem is that the Pareto-optimal region is nonconvex. Thus, weighted approaches will have difficulty in finding a good spread of solutions on the Pareto-optimal front.

ZDT3

This is an $n = 30$ variable problem having a number of disconnected Pareto-optimal fronts:

$$\text{ZDT3:} \begin{cases} f_1(\mathbf{x}) &= x_1, \\ g(\mathbf{x}) &= 1 + \frac{9}{n-1} \sum_{i=2}^{n} x_i, \\ h(f_1, g) &= 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1). \end{cases} \tag{8.35}$$

All variables lie in the range $[0, 1]$. The Pareto-optimal region corresponds to $x_i^* = 0$

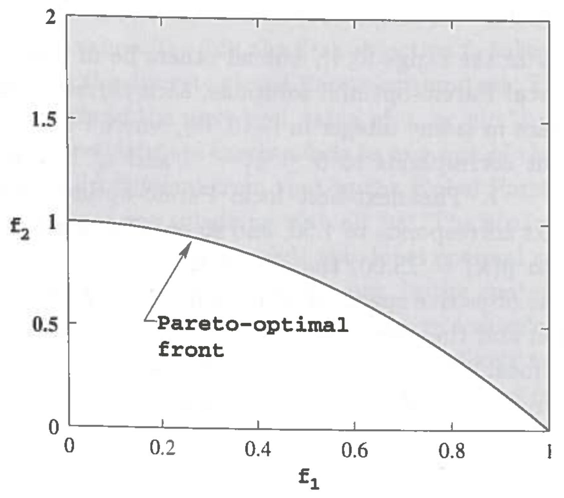

Figure 214 The search space near the Pareto-optimal region for ZDT2.

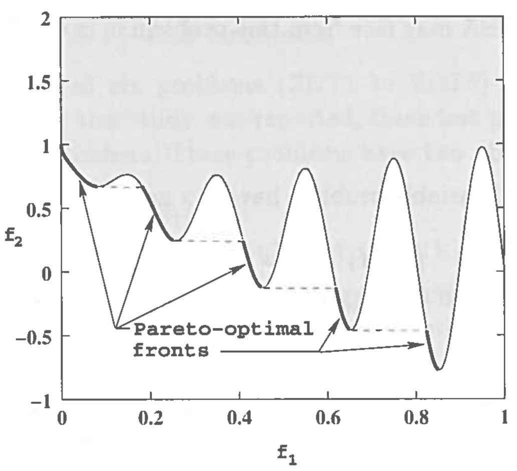

Figure 215 The search space near the Pareto-optimal region for ZDT3.

for $i = 2, 3, \ldots, 30$, and hence not all points satisfying $0 \le x_1^* \le 1$ lie on the Pareto-optimal front. Figure 215 shows the search region and the Pareto-optimal fronts in the objective space. The difficulty with this problem is that the Pareto-optimal region is discontinuous. The real test for MOEAs would be to find all discontinuous regions with a uniform spread of non-dominated solutions.

ZDT4

This is an $n = 10$ variable problem having a convex Pareto-optimal set:

$$\text{ZDT4:} \begin{cases} f_1(x) &= x_1, \\ g(x) &= 1 + 10(n-1) + \sum_{i=2}^{n} (x_i^2 - 10\cos(4\pi x_i)), \\ h(f_1, g) &= 1 - \sqrt{f_1/g}. \end{cases} \qquad (8.36)$$

The variable x_1 lies in the range $[0, 1]$, but all others lie in $[-5, 5]$. There exists 21^9 or about $8(10^{11})$ local Pareto-optimal solutions, each corresponding to $0 \le x_1^* \le 1$ and $x_i^* = 0.5m$, where m is any integer in $[-10, 10]$, where $i = 2, 3, \ldots, 10$. The global Pareto-optimal front corresponds to $0 \le x_1^* \le 1$ and $x_i^* = 0$ for $i = 2, 3, \ldots, 10$. This makes $g(x^*) = 1$. The next-best local Pareto-optimal front corresponds to $g(x) = 1.25$, the next corresponds to 1.50, and so on. The worst local Pareto-optimal front corresponds to $g(x) = 25.00$, thereby making a total of 100 distinct Pareto-optimal fronts in the objective space, of which only one is global. Figure 216 shows a partial search region and the Pareto-optimal front in the objective space. The sheer number of multiple local Pareto-optimal fronts produces a large number of hurdles for an MOEA to converge to the global Pareto-optimal front.

ZDT5

This is a Boolean function defined over bit-strings. There are a total of 11 discrete variables. The first variable x_1 is presented by a 30-bit substring and the rest 10