CHEPTER IV

EVOLUTIONNARY ALGORITHMS

4.1 Overview

Evolutionary algorithms are stochastic search methods that mimic the metaphor of natural biological evolution. Evolutionary algorithms operate on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain an d breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation.

Evolutionary algorithms model natural processes, such as selection, recombination, mutation, migration, locality and neighborhoods. Fig. 4.1 shows the structure of a simple genetic algorithm. Evolutionary algorithms work on populations of individuals instead of single solutions. In this way the search is performed in a parallel manner.

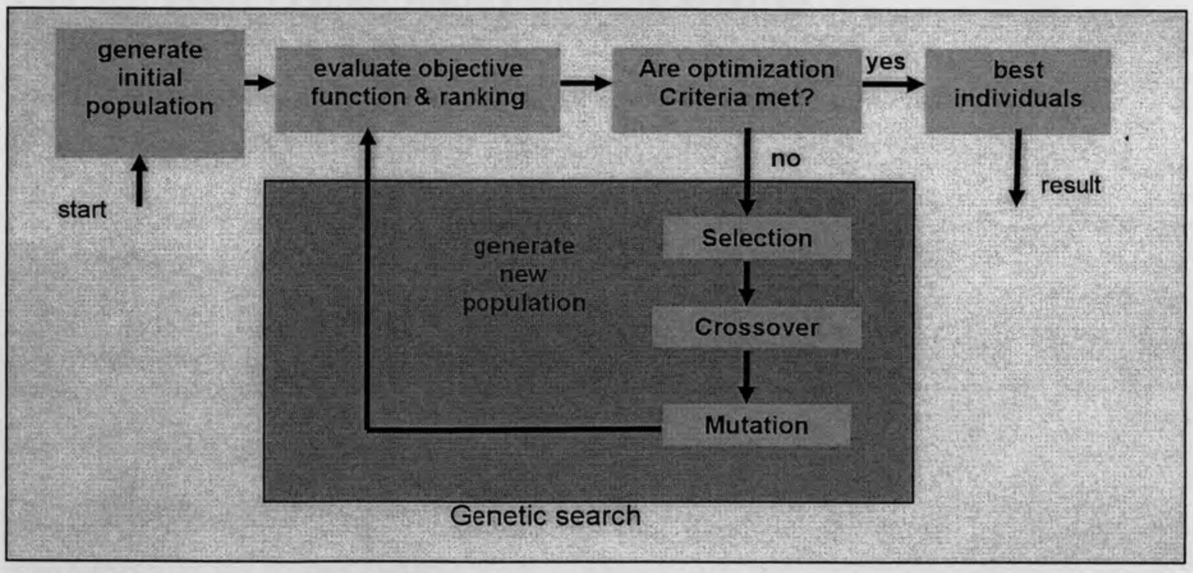

Figure 4.1: Structure of a single population evolutionary algorithm

At the beginning of the computation a number of individuals (the population) are randomly initialized. The objective function is then evaluated for these individuals. The first/initial generation is produced.

If the optimization criteria are not met the creation of a new generation starts. Individuals are selected according to their fitness for the production of offspring. Parents are recombined to produce offspring. All offspring will be mutated with a certain probability. The fitness of the offspring is then computed. The offspring are inserted into the population replacing the parents, producing a new generation. This cycle is performed until the optimization criteria are reached.

Such a single population evolutionary algorithm is powerful and performs well on a broad class of problems. However, better results can be obtained by introducing many populations, called subpopulations. Every subpopulation evolves for a few generations isolated (like the single population evolutionary algorithm) before one or more individuals are exchanged between the subpopulations. The Multi-population evolutionary algorithm models the evolution of a species in a way more similar to nature than the single population evolutionary algorithm.

From the above discussion, it can be seen that evolutionary algorithms differ substantially from more traditional search and optimization methods. The most significant differences are:

-Evolutionary algorithms search a population of points in parallel, not a single point.

-Evolutionary algorithms do not require derivative information or other auxiliary knowledge; only the objective function and corresponding fitness levels influence the directions of search.

-Evolutionary algorithms use probabilistic transition rules, not deterministic ones.

-Evolutionary algorithms are generally more straightforward to apply

-Evolutionary algorithms can provide a number of potential solutions to a given problem.

The final choice is left to the user. (Thus, in cases where the particular problem does not have one individual solution, for example a family of Pareto-optimal solutions, as in the case of multi-objective optimization and scheduling problems, then the evolutionary algorithm is potentially useful for identifying these alternative solutions simultaneously).

4.2 Basic principles of Evolutionary algorithms

In general, an EA is characterized by three facts:

1. A set of solution candidates is maintained, which
2. Undergoes a selection process and
3. Is manipulated by genetic operators, usually recombination and mutation.

By analogy to natural evolution, the solution candidates are called *individuals* and the set of solution candidates is called the *population*. Each individual represents a possible solution, i.e., a decision vector; however, an individual *is not* a decision vector but rather encodes it based on an appropriate structure. Without loss of generality, this structure is assumed to be a vector here, e.g., a bit vector or a real-valued vector, although other structures like trees (Koza, 1992) might be used as well; the set of all possible vectors constitutes the *individual space I*. In this terminology, the population is a set of vector $i \in I$, to be more precise a multi-set of vectors since it can contain several identical individuals.

In the selection process, which can be either stochastic or completely deterministic, low-quality individuals are removed from the population, while high quality individuals are reproduced. The goal is to focus the search on particular portions of the search space and to increase the average quality within the population. The quality of an individual with respect to the optimization task is represented by a scalar value, the so-called *fitness*. Note that since the quality is related to the objective functions and the constraints, an individual must first be decoded before its fitness can be calculated. This situation is illustrated in Fig. 4.2 Given an individual $i \in I$. A mapping function m encapsulates the decoding algorithm to derive the decision vector $x = m(i)$ from i. Applying f to x yields the corresponding objective vector on the basis of which a fitness value is assigned to i.

Recombination and mutation aim at generating new solutions within the search space by the variation of existing ones. The crossover operator takes a certain number of parents and creates a certain number of children by recombining the parents. To mimic the stochastic nature of evolution, a crossover probability is associated with this operator. By contrast, the mutation operator modifies individuals by changing small parts in the associated vectors according to a given mutation rate. Both crossover and mutation operator work on individuals,i.e., in individual space and not on the decoded decision vectors. Based on the above concepts, natural evolution is simulated by an iterative computation process. First, an initial population is created at random (or according to a predefined scheme), which is the starting point of the evolution process. Then a loop consisting of the steps evaluation (fitness assignment), selection, recombination, and/or mutation is executed a certain number of times. Each loop iteration is called a *generation*, and often a predefined maximum number of generations serves as the termination criterion of the loop. But also other conditions, e.g., stagnation in the population or existence of an individual with sufficient quality, may be used to stop the simulation. At the end, the best individual(s) in the final population or found during the entire evolution process is the outcome of the EA. In the following, the basic structure of an EA, as it is used throughout this work, is formalized. The population *P* at a certain generation *t* is represented by the symbol P_t , and the symbol + stands for multi-set union in conjunction with populations.

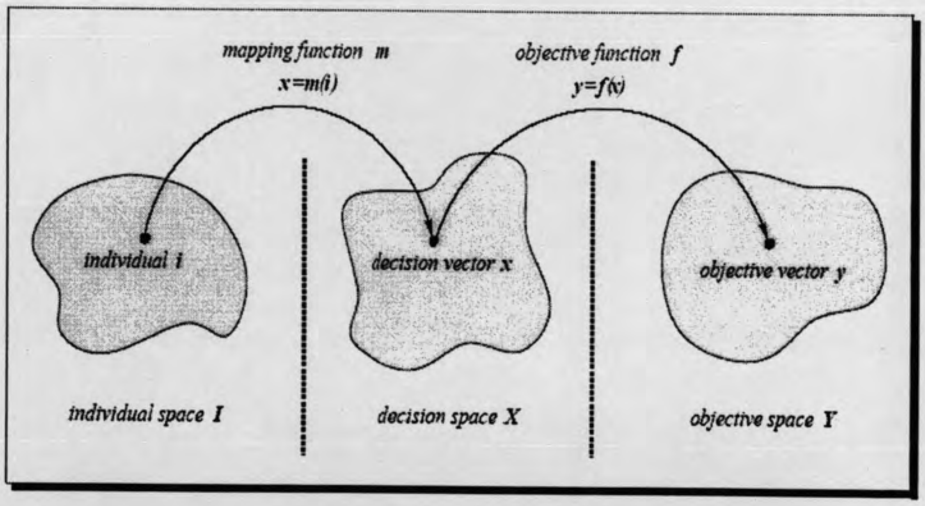

Figure 4.2: Relation between individual space, decision space, and objective space.

Alg. 1: (General Evolutionary Algorithm)

Input: N (population size)

T (maximum number of generations)

Pc (crossover probability)

Pm (mutation probability)

```
Output: $\mathbf{A}$ (non-dominated set)

Step 1: Initialization: Set $P_0=\emptyset$ and $t = 0$. For $i = 1, \ldots, N$ do
        a) Choose $i \in \mathbf{I}$ according to some probability distribution.
        b) Set $P_0 = P_0 + \{i\}$.
Step 2: Fitness assignment: For each individual $i \in P_t$ determine the encoded decision vector $x = m(i)$ as well as the objective vector $y = f(x)$ and calculate the scalar fitness value $F(i)$.
Step 3: Selection: Set $P' = \emptyset$. For $i = 1, \ldots, N$ do
        a) Select one individual $i \in P_t$ according to a given scheme and based on its fitness value $F(i)$.
        b) Set $P' = P' + \{i\}$.
        The temporary population P' is called the mating pool.
Step 4: Recombination: Set $P'' = \emptyset$. For $i = 1, \ldots, N/2$ do
        a) Choose two individuals $i, j \in P'$ and remove them from $P'$.
        b) Recombine $i$ and $j$. The resulting children are $k, l \in \mathbf{I}$.
        c) Add $k, l$ to $P''$ with probability Pc. Otherwise add $i, j$ to $P''$.
Step 5: Mutation: Set $P''' = \emptyset$. For each individual $i \in P''$ do
        a) Mutate $i$ with mutation rate Pm. The resulting individual is $j \in \mathbf{I}$.
        b) Set $P''' = P''' + \{j\}$.
Step 6: Termination: Set $P_{t+1} = P'''$ and $t = t + 1$. If $t \geq T$ or another stopping criterion is satisfied then set $A = p(m(P_t))$ else go to Step 2.
```

It must be emphasized that this algorithm does not reflect an EA in its most general form as, e.g., the population size need not be restricted and recombination can also involve more than two parents. Moreover, a large number of selection, crossover, and mutation operators have been proposed for different representations, applications, etc. which, however, are not presented here. A thorough discussion of the various aspects of EAs can be found in the following standard introductory material (Goldberg 1989; Koza 1992; Fogel 1995; Back 1996; Mitchell 1996).

4.3 Key issues in multi-objective search

As mentioned at the beginning of this chapter, with an MOP the optimization goal it consists of multiple objectives. Two major problems must be addressed when an EA is applied to multi-objective optimization:

1. How to accomplish fitness assignment and selection, respectively, in order to guide the search towards the Pareto-optimal set.

2. How to maintain a diverse population in order to prevent premature convergence and achieve a well distributed and well spread non-dominated set.

In the following, a categorization of general techniques which deal with these issues is presented. The focus here is on cooperative population searches where only one optimization run is performed in order to approximate the Pareto optimal set. Moreover, another issue, elitism, is briefly discussed since it is different and more complicated with Multi-Objective Optimization Problems (MOPs) than with Single- Multi-Objective Optimization Problems (SOPs).

4.3.1 Fitness Assignment and Selection

In contrast to single-objective optimization, where objective function and fitness function are often identical, both fitness assignment and selection must allow for several objectives with Multi-objective optimization problems. In general, one can distinguish Multi-objective Evolutionary algorithms where the objectives are considered separately, approaches that are based on the classical aggregation techniques, and methods which make direct use of the concept of Pareto dominance.

4.3.1.1 Selection by Switching Objectives

Instead of combining the objectives into a single scalar fitness value, this class of Multi-objective Evolutionary algorithms switches between the objectives during the selection phase. Each time an individual is chosen for reproduction, potentially a different objective will decide which member of the population will be copied into the mating pool. As a consequence, Steps 2 and 3 of the general EA are usually integrated or executed alternately. For example, Schaffer (1985) proposed filling equal portions of the mating pool according to the distinct objectives, while Fourman (1985) implemented a selection scheme where individuals are compared with regard to a specific (or random) order of the objectives. Later, Kursawe (1991) suggested assigning a probability to each objective which determines whether the objective will be the sorting criterion in the next selection step—the probabilities can be user-defined or chosen randomly over time. All of these approaches may have a bias towards "extreme" solutions and be sensitive to non-convex Pareto-optimal fronts (Horn 1997).

4.3.1.2 Aggregation Selection with Parameter Variation

Other MOEA implementations build on the traditional techniques for generating trade-off surfaces. As with these methods, the objectives are aggregated into a single parameterized objective function; however, the parameters of this function are not changed for different optimization runs, but instead systematically varied during the same run. Some approaches (Hajela and Lin 1992) (Ishibuchi and Murata 1996), for instance, use the weighting method. Since each individual is assessed using a particular weight combination (either encoded in the individual or chosen at random), all members of the population are evaluated by a different objective function. Hence, optimization is done in multiple directions simultaneously. Nevertheless, the potential disadvantages of the underlying scalarization method (e.g., a bias towards convex portions of the Pareto-optimal front) may restrict the effectiveness of such Multi-objective Evolutionary algorithms (Veldhuizen 1999).

4.3.1.3 Pareto-based Selection

The concept of calculating an individual's fitness on the basis of Pareto dominance was first suggested by Goldberg (1989). He presented a "revolutionary 10-line sketch" (Deb 1999b) of an iterative ranking procedure: First all non- dominated individuals are assigned rank one and temporarily removed from the population. Then, the next non-dominated individuals are assigned rank two and so forth. Finally, the rank of an individual determines its fitness value. Remarkable here is the fact that fitness is related to the whole population, while with other aggregation techniques an individual's raw fitness value is calculated independently of other individuals. This idea has been taken up by numerous researchers, resulting in several Pareto-based fitness assignment schemes, e.g, (Fonseca and Fleming 1993; Horn, Nafpliotis, and Goldberg 1994; Srinivas and Deb 1994). Although this class of Multi-objective Evolutionary algorithms is theoretically capable of finding any Pareto-optimal solution, the dimensionality of the search space may influence its performance, as noted by Fonseca and Fleming (1995):

True Pareto EAs cannot be expected to perform well on problems involving many competing objectives and may simply fail to produce satisfactory solutions due to the large dimensionality and the size of the trade-off surface." Even so, Pareto-based techniques seem to be most popular in the field of evolutionary multi-objective optimization (Veldhuizen 1998).

4.3.2 Population Diversity

In order to approximate the Pareto-optimal set in a single optimization run, evolutionary optimizers have to perform a multimodal search where multiple, widely different solutions are to be found. Therefore, maintaining a diverse population is crucial for the efficacy of an MOEA. Unfortunately, a simple (elitist) EA tends to converge towards a single solution and often loses solutions due to three effects (Mahfoud 1995): selection pressure, selection noise, and operator disruption. The selection pressure is often defined in terms of the takeover time, i.e., the time required until the population is completely filled by the best individual when only selection takes place (Back 1996). Selection noise refers to the variance of a selection scheme, while operator disruption relates to the destructive effects which recombination and mutation may have (e.g., high-quality individuals may be disrupted). To overcome this problem, several methods have been developed; the ones most frequently used in evolutionary multi-objective optimization are briefly summarized here.

4.3.2.1 Fitness Sharing

Fitness sharing (Goldberg and Richardson 1987), which is the most frequently, used technique, aims at promoting the formulation and maintenance of stable subpopulations (*niches*). It is based on the idea that individuals in a particular niche have to share the available resources. The more individuals are located in the neighborhood of a certain individual, the more its fitness value is degraded. The neighborhood is defined in terms of a distance measure ***d(i , j)*** and specified by the so-called *niche radius* $\sigma_{share\ e}$. Mathematically, the shared fitness ***F(i)*** of an individual $i \in P$ is equal to its old fitness $F^{'}(i)$ divided by its *niche count*:

$$F(i) = \frac{F'(i)}{\sum_{j \in P} s(d(i,j))} \tag{4.1}$$

An individual's niche count is the sum of *sharing function* (*s*) values between itself and the individuals in the population. A commonly-used sharing function is

$$s(d(i\ ,j\ )) = \begin{cases} 1-(\frac{d(i,j)}{\sigma\text{share}})^{\alpha} & \text{if } d(i\ ,j) < \sigma share \\ 0 & \textbf{otherwise} \end{cases} \tag{4.2}$$

Furthermore, depending on how the distance function *d(i , j)* is defined, one distinguishes three types of sharing:

1. Fitness sharing in individual space (***d(i , j) = ||i – j||***).
2. Fitness sharing in decision space (***d(i , j) = ||m(i) - m(j)||***), and
3. Fitness sharing in objective space (***d(i , j) = ||f (m(i)) - f (m(j))||***).

Where ||·|| denotes an appropriate distance metric. Currently, most MOEAs implement fitness sharing, e.g., (Hajela and Lin 1992; Fonseca and Fleming 1993; Horn, Nafpliotis, and Goldberg 1994; Srinivas and Deb 1994; Greenwood, Hu, and D'Ambrosio 1996; Todd and Sen 1997; Cunha, Oliviera, and Covas 1997). 4.3.2.1 Restricted mating basically, two individuals is allowed to mate only if they are within a certain distance of each other (given by the parameter $\sigma_{share\ e}$). As with fitness sharing, the distance of two individuals can be defined in individual space, decision space, or objective space. This mechanism may avoid the formation of lethal individuals and therefore improve the online performance. Nevertheless, as mentioned in (Fonseca and Fleming 1995b), it does not appear to be widespread in the field of MOEAs, e.g., (Hajela and Lin 1992; Fonseca and Fleming 1993; Loughlin and Ranjithan 1997).

4.3.2.2 Isolation by Distance

This type of diversity mechanism assigns each individual a location (Ryan 1995) where in general two approaches can be distinguished. Either a spatial structure is defined on one population such that spatial niches can evolve in the same population, or there are several distinct populations which only occasionally exchange individuals (migration). Poloni (1995), for instance, used a distributed EA with multiple small populations, while Laumanns, Rudolph, and Schwefel (1998) structured the population by an underlying graph, a two-dimensional torus, where each individual is associated with a different node.

4.3.2.3 Over specification

With this method, the individual contains active and inactive parts: the former specify the encoded decision vector, the latter are redundant and have no function. Since inactive parts can become active and vice versa during the evolution, information can be hidden in an individual. Diploidy (Goldberg 1989) is an example of over specification that is used in the MOEA proposed by Kursawe (1991).

4.3.2.4 Reinitialization

Another technique to prevent premature convergence is to reinitialize the whole or parts of the population after a certain period of time or whenever the search stagnates. For example, Fonseca and Fleming (1998a) presented a unified formulation of evolutionary multi-objective optimization where at each generation a small number of random immigrants is introduced in the population.

4.3.2.5 Crowding

Finally, *crowding* (De Jong 1975) and its derivates seem to be rather seldomly implemented in MOEAs, e.g., (Blickle 1996). Here, new individuals (children) replace similar individuals in the population. In contrast to Algorithm 1, not the whole population is processed by selection, recombination, and mutation, but only a few individuals are considered at a time.

4.3.3 Elitism

De Jong (1975) suggested a policy to always include the best individual of $\boldsymbol{P_t}$ into $\boldsymbol{P_{t+1}}$ in order to prevent losing it due to sampling effects or operator disruption. This strategy, which can be extended to copy the *b* best solutions to the next generation, is denoted as *elitism*. In his experiments, De Jong found that elitism can improve the performance of a genetic algorithm on unimodal functions, while with multimodal functions it may cause premature convergence. In evolutionary multi-objective optimization, elitism plays an important role, as will be shown in the next chapter. As opposed to single-objective optimization, the incorporation of elitism in MOEAs is substantially more complex. Instead of one best individual, there is an elite set whose size can be considerable compared to the population, for instance, when the Pareto-optimal set contains an infinite number of solutions. This fact involves two questions which must be answered in this context:

• **Population $\rightarrow$ Elite Set:**

Which individuals are kept for how long in the elite set?

• **Elite Set $\rightarrow$ Population:**

When and how is which members of the elite set re-inserted into the population?

In general, two basic elitist approaches can be found in MOEA literature. One strategy, hich directly uses De Jong's idea, is to copy those individuals from $\boldsymbol{P_t}$ automatically to *Pt*+1 whose encoded decision vectors are non-dominated regarding $\boldsymbol{m(P_t)}$ (Tamaki, Mori, Araki, Mishima, and Ogai 1994). Sometimes a more restricted variant is implemented where only the *k* individuals whose corresponding objective vectors maximize one of the *k* objectives constitute the elite set (Anderson and Lawrence 1996; Murata, Ishibuchi, and Tanaka 1996; Todd and Sen 1997). Also the so-called $(\lambda + \mu)$ selection mainly used in the area of evolutionary strategies (Back 1996) belongs to this class of elitist strategies. For example, Rudolph (1998) examined a simplified version of the MOEA originally presented in (Kursawe, 1991) which is based on (1+1) selection. Often used is also the concept of maintaining an external set *P* of individuals whose encoded decision vectors are non-dominated among all solutions generated so far. In each generation, a certain percentage of the population is filled up or replaced by members of the external set these members are either selected at random (Cieniawski, Eheart, and Ranjithan 1995; Ishibuchi and Murata 1996) or according to other criteria, such as the period that an individual has stayed in the set (Parks and Miller 1998). Since the external set may be considerably larger than the population, Parks and Miller (1998) only allow those individuals to be copied into the elite set which are sufficiently dissimilar to the existing elite set members. Occasionally, both of the above two elitist policies are applied (Murata, Ishibuchi, and Tanaka 1996; Todd and Sen 1997).

4.4 Overview of Evolutionary Techniques in thesis

4.4.1 Multiple Objective Genetic Algorithm (MOGA): a brief overview

Fonseca and Fleming (1993) first introduced a multi-objective GA (called MOGA) which used the non-dominated classification of a GA population. The investigators were the first to suggest a multi-objective GA which explicitly caters to emphasize non-dominated solutions simultaneously maintains diversity in the non- dominated solutions.

First, each solution is checked for its domination in the population. To a solution i, a rank equal to one plus the number of solution n_i that dominate solution i is assigned

$$r_i = 1 + n_i \tag{4.3}$$

In this way, non-dominated solutions are assigned a rank equal to 1, since no solution would dominate a non-dominated solution in a population. A little thought reveals that in any population, there must be at least one solution with rank equal to one and the maximum rank of any population member cannot be more than N (the population size). Once the ranking is performed, a raw fitness to a solution is assigned based on its rank. To perform this, first the ranks are sorted in ascending order of magnitude. Then, a raw fitness is assigned to each solution by using a linear (or any other) mapping function. Thereafter, solutions of each rank are considered at a time and their raw finesses are averaged. This average fitness is called the assigned fitness to each solution of the rank. In this way, the total allocated raw fitness and total assigned fitness to each rank remains identical. Moreover, the mapping and averaging procedure ensure that the better ranked solutions have a higher assigned fitness. In this way, non-dominated solutions are emphasized in a population.

In order to maintain diversity among non-dominated solutions Fonseca and Fleming (1993) have introduced niching among solutions of each rank. The niche count with σ_{share} is described in the previous. The sharing function with $\alpha = 1$ is used, but the distance metric is computed with the objective function values, instead of the parameter values. Thus, the normalized distance between any two solutions i and j in a rank is calculated as follows:

$$d_{ij} = \sqrt{\sum_{k=1}^{M} (\frac{f_k^{(i)} - f_k^{(j)}}{f_k^{\max} - f_k^{\min}})^2} \tag{4.4}$$

Where $f_k^{\max}$ **and** $f_k^{\min}$ are the maximum and minimum objective function value of the k-th objective. For the solution i , d_{ij} is computed for each solution j (including i) having the same rank . Thereafter, the niche count is calculated by summing the sharing function values:

$$nc_i = \sum_{j=1}^{u(ri)} sh(d_{ij}) \tag{4.5}$$

Where $u(r_i)$ is the number of solution in rank r_i. In an MOGA, the sharing function approach, as described earlier in section 4.3.2.1 is followed and a shared fitness value is calculated by dividing the fitness of the solution by its niche count. Although all solution of any particular rank has the identical fitness, the shared fitness value of a solution residing in a less-crowed region has a better shared fitness. This produces a large selection pressure for poorly represented solutions in any rank.

Dividing the assigned fitness values by the niche count (always equal to or greater than one) reduces the fitness of each solution. In order to keep the average fitness of the solutions in a rank the same as that before sharing, these fitness values are scaled so that their average shared fitness value is the same as the average assigned fitness value. After these calculations, the focus is shifted to the solutions of the next rank processed. Thereafter, the stochastic universal selection, the single-point crossover, and the bit-wise mutation operators are applied to create a new population.

In Fig. 4.3, a hypothetical population and the corresponding ranks of the individuals are shown. The individuals whose associated solutions are non-dominated regarding m(P) have rank 1 while the worst individual was assigned rank 10. Based on the ranks, the mating pool is filled using stochastic universal sampling (Baker 1987).

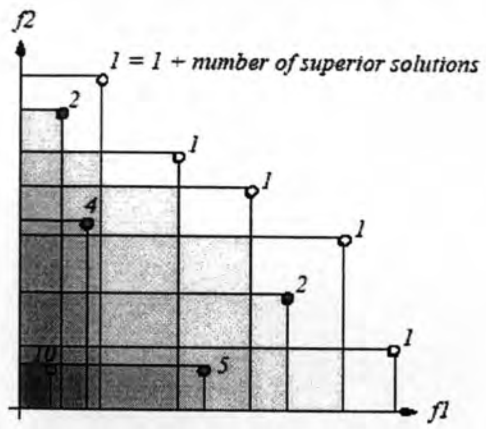

Figure 4.3: Ranking procedure in the MOGA (maximum f1 and maximum f2).

4.4.2 Niched Pareto Genetic Algorithm (NPGA): a brief overview

Horn et al. proposed a tournament selection scheme based on Pareto dominance principles. Only two individuals are randomly selected for tournament. To find the winner solution, a comparison set that contains a number of other individuals in the population is randomly selected. Then, the dominance of both candidates with respect to the comparison set is tested. If one candidate only dominates the comparison set, he is selected as the winner. Otherwise, implement sharing procedure to specify the winner candidate. Generally, the tournament selection is carried out as follows.

Pareto domination tournaments. Consider a set of N population members, each having N_{obj} objective function values. The following procedure can be used to find the non-dominated set of solutions:

1. Begin with i = 1.
2. Pick randomly two candidates for selection x^1 and x^2.
3. Pick randomly a comparison set of individuals from the population.
4. Compare each candidate, x^1 and x^2, against each individual in the comparison set for domination using the conditions for domination.
5. If one candidate is dominated by the comparison set while the other is not, then select the later for reproduction and go to Step 7, else proceed to Step 6.

6. If neither or both candidates are dominated by the comparison set, then use sharing to choose the winner.

7. If i = N is reached, stop selection procedure, else set i = i + 1 and go to Step 2.

Sharing procedure. To prevent the genetic drift problem, a form of sharing should be carried out when there is no preference between two candidates. This form of sharing maintains the genetic diversity along the population fronts and allows the GA to develop a reasonable representation of the Pareto-optimal front. Generally, the basic idea behind sharing is: the more individuals are located in the neighborhood of a certain individual, the more its fitness value is degraded.

The sharing procedure is performed in the following way for the candidate i:

1. Begin with j =1:

2. Compute a normalized Euclidean distance measure with another individual j in the current population, as follows

$$d_{ij} = \sqrt{\sum_{k=1}^{M} \left(\frac{f_k^{(i)} - f_k^{(j)}}{f_k^{\max} - f_k^{\min}}\right)^2} \qquad (4.4)$$

Where N_{obj} is the number of problem objectives. The parameters $f_k^{\max}$ and $f_k^{\min}$ are the upper and lower values of the *k*-th objective function J_k.

3. This distance d_{ij} is compared with a prespecified niche radius σshare and the following sharing function value is computed as:

$$\mathrm{Sh}(d_{ij}) = \begin{cases} 1 - \left(\dfrac{d_{ij}}{\sigma\text{share}}\right)^2, \text{if} d_{ij} \le \sigma\text{share} \\ 0, \text{otherwise} \end{cases} \qquad (4.2)$$

4. Set j = j + 1. If j ≤ N; go to Step 2, else calculate niche count for the candidate i as follows: $m_i = \sum_{j=1}^{N} \mathrm{Sh}(d_{ij})$

5. Repeat the above steps for the second candidate.

6. Compare m_1 and m_2. If $m_1 < m_2$ then choose the first candidate, else choose the second candidate.

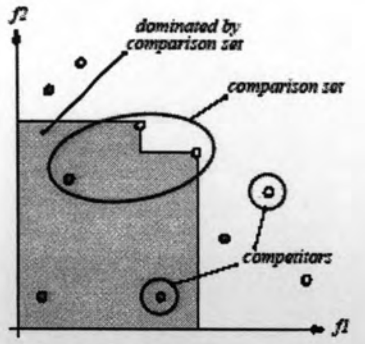

Figure 4.4: Ranking procedure in the NPGA (maximum f1 and maximum f2).

The mechanism of the binary Pareto tournaments is illustrated in Fig. 4.4, two competing individuals and a set of *tdom* individuals are compared. The competitor represented by the white point is the winner of the tournament since the encoded decision vector is not dominated with regard to the comparison set in contrast to the other competitor.

4.4.3 Non-dominated Sorting Genetic Algorithm (NSGA): a brief overview

Goldberg's idea of using the non-dominated concept in GAs (Goldberg, 1989) was more directly implemented by Srinivas and Dep in 1994. Once again, the dual objectives in multi-objective optimization algorithm are maintained by using a fitness assignment scheme which prefers non-dominated solutions and by using a sharing strategy which preserves diversity among solutions of each non-dominated front.

The first step of an NSGA is to sort the population P according to non-domination. This classifies the population into a number of mutually exclusive equivalent classes (or non-dominated sets) P_j:

$$P = U_{j=1}^{p} P_j \qquad (4.6)$$

Once the classification task is over, it is clear that all solutions in the first set, that is all $i \in P_1$, belong to the best non-dominated set in the population. The second best solutions in the population are those that belong to second set, or all members of P_2, and so on. Obviously, the worst solutions are those belonging to the final set, or the members of P_p. Where *p* is the number of different non-dominated sets in the population. It is clear that solutions of the first front are best in terms of their closeness to the true Pareto-optimal front in the population. Thus,

123500918

it makes sense to assign the best non-dominated front and then assign a progressively worse fitness to solutions of higher fronts. This is because the best non-dominated solutions in a population are nearest to the true Pareto-optimal front compared to any other solution in the population.

The fitness assignment procedure begins from the first non-dominated set and the successively proceeds to dominated sets. Any solution i of the first dominated set is assigned a fitness equal to F_i =N (the population size). The sharing function method is used is used front-wise. That is, for each solution i in the front P_1, the normalized Euclidean distance d_{ij} from another solution j in the same front is calculated, as follows:

$$d_{ij} = \sqrt{\sum_{k=1}^{M} (\frac{x_k^{(i)} - x_k^{(j)}}{x_k^{\max} - x_k^{\min}})^2} \quad (4.7)$$

The sharing function takes a value between zero and one, depending on the distance d_{ij}. Any solution j which has a distance greater than σshare from i-th solution contributes noting to the sharing function value, After all | P_1| sharing function values are calculated, they are added together to calculate the niche count nc_i of the i-th solution. The niche count, in some sense, denotes the number of solution in neighborhood of i-th solution, including the latter. If there exists no other solution within a radius of σshare from the solution, the niche count for that solution would be one. The final task is to reduce the fitness of the i-th solution by its niche count and obtain the shared fitness value, $F_i^{'} = F_i / nc_i$ .This process of degrading fitness of a solution which crowded by many solutions helps emphasize the solutions residing in less crowed region.

Since the sharing function method works with the proportionate selection operator, the NSGA used the less-noisy stochastic remainder roulette-wheel operator (Goldberg, 1989).This assigns copies in the mating pool proportional to the shared fitness. In this way, each solution in the first front has a better chance of surviving in the mating pool than that the second front, and so on. On the other hand, the sharing function approach among solution in each front makes sure that solutions in the less crowded region get more copies in the mating pool. As mentioned above, the crossover and mutation operators are applied as usual to the whole population.

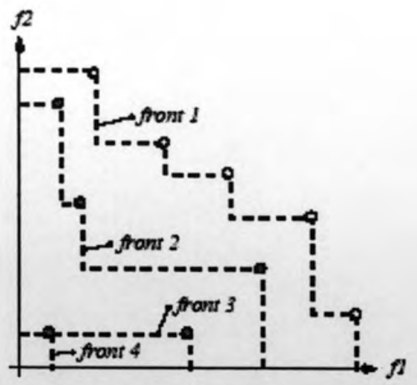

Figure 4.5: Ranking procedure in the NSGA (maximum f1 and maximum f2).

4.4.4 Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II): a brief overview

The non-dominated sorting GA (NSGA) proposed by Srinivas and Deb in 1994 has been applied to various problems. However as mentioned earlier there have been a number of criticisms of the NSGA. In this section, we modify the NSGA approach in order to alleviate all the above difficulties. We begin by presenting a number of different modules that form part of NSGA-II.

4.4.4.1 A fast non-dominated sorting approach

In order to sort a population of size N according to the level of non-domination, each solution must be compared with every other solution in the population to find if it is dominated. This requires $O(mN)$ comparisons for each solution, where m is the number of objectives. When this process is continued to find the members of the first non-dominated class for all population members, the total complexity is $O(mN^2)$. At this stage, all individuals in the first non-dominated front are found. In order to find the individuals in the next front, the solutions of the first front are temporarily discounted and the above procedure is repeated. In the worst case, the task of finding of the second front also requires $O(mN^2)$ computations. The procedure is repeated to find the subsequent fronts. As can be seen the worst case (when there exists only one solution in each front) complexity of this algorithm is $O(mN^3)$. In the following we describe a fast non-dominated sorting approach which will require at most computations.

First, for each solution we calculate two entities: (i) n_i, the number of solutions which dominate the solution i, and (ii) S_i, a set of solutions which the solution i dominates. The calculation of these two entities requires O(mN^2) comparisons. We identify all those points which have $n_i = 0$ and put them in a list F_1. We call F_1 the current front. Now, for each solution in the current front we visit each member (j) in its set S_i and reduce its n_j count by one. In doing so, if for any member j the count becomes zero, we put it in a separate list F_1. When all members of the current front have been checked, we declare the members in the list F_1 as members of the first front. We then continue this process using the newly identified front H as our current front.

Each such iteration requires O(N) computations. This process continues till all fronts are identified. Since at most there can be N fronts, the worst case complexity of this loop is O(N^2). The overall complexity of the algorithm now is O(mN^2) + O(N^2) or O(mN^2).It is worth mentioning here that although the computational burden has reduced from O(mN^3) to O(mN^2) by performing systematic book-keeping, the storage has increased from O(N) to O(N^2) in the worst case.

The fast non-dominated sorting procedure which when applied on a population returns a list of the non-dominated fronts F.

fast non-dominated sort (P)

```
    for each $p \in P$
     for each $q \in P$
    if $(p \prec q)$ then                    if $p$ dominates $q$ then
          $S_P = S_P \cup \{q\}$              include q in $S_P$
    else $(q \prec p)$ then                  if $p$ is dominated by $q$ then
    $n_p = n_P + 1$                          increment $n_p$
    if $n_p$=0 then                          if no solution dominates $p$ then
    $F_1 = F_1 \cup \{p\}$                   $p$ is member of first front
    $i = 1$
    while $F_i = \phi$
          $H = \phi$
    for each $p \in F_i$                     for each member $p$ in $F_i$
    for each $q \in S_p$                     modify each member from the set $S_p$
```

```
        $n_q = n_q - 1$                      decrement $n_q$ by one
if $n_q = 0$ then $H = H \cup \{q\}$  if $n_q$ is zero, q is member of list $H$
$i = i + 1$
$F_i = H$                                current front is formed with all members of $H$
```

4.4.4.2 Density Estimation

To get an estimate of the density of solutions surrounding a particular point in the population we take the average distance of the two points on either side of this point long each of the objectives. This quantity $i_{distance}$ serves as an estimate of the size of the largest cuboids enclosing the point i without including any other point in the population (we call this the *crowding distance*). In Fig. 4.6, the crowding distance of the i-th solution in its front (marked with solid circles) is the average side-length of the cuboids (shown with a dashed box). The following algorithm is used to calculate the crowding

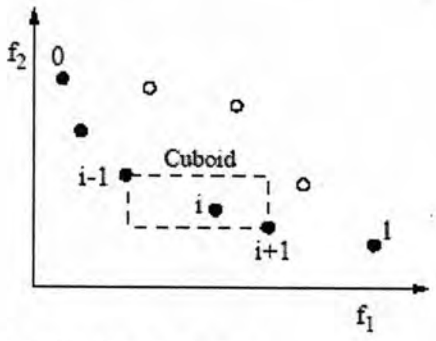

Figure 4.6: The crowding distance calculation is shown.

distance of each point in the set I:

```
$l = |I|$                                          number of solutions in $I$
for each $i$, set $I(i)_{distance} = 0$            initialize distance
for each objective $m$
$I = sort(I, m)$                                   sort using each objective value
$I(1)_{distance} = I(l)_{distance}$                so that boundary points are always selected
for $i = 2$ to $(l-1) = \infty$                    for all other points
$I(i)_{distance} = I(i)_{distance} + (I(i+1).m - I(i-1).m)$
```

Here refers $I(i).m$ to the m -th objective function value of the i -th individual in the set. The complexity of this procedure is governed by the sorting algorithm. In the worst case (when all solutions are in one front), the sorting requires O(*mN log N*) computations.

4.4.4.3 Crowded Comparison Operator

The crowded comparison operator $\prec_n$ guides the selection process at the various stages of the algorithm towards a uniformly spread out Pareto-optimal front. Let us assume that every individual i in the population has two attributes.

1. Non-domination rank (i_{rank})
2. Local crowding distance ($i_{dis\tan ce}$)

We now define a partial order as $\geq_n$:

$$i \prec_n j \quad if(i_{rank} < j_{rank})or((i_{rank} = j_{rank})and(i_{dis\tan ce} > j_{dis\tan ce}))$$

That is, between two solutions with differing non-domination ranks we prefer the point with the lower rank. Otherwise, if both the points belong to the same fronts then we prefer the point which is located in a region with lesser number of points (the size of the cuboids inclosing it is larger).

4.4.4.4 The Main Loop

Initially, a random parent population is created. The population is sorted based on the non-domination. Each solution is assigned a fitness equal to its non-domination level (1 is the best level). Thus, minimization of fitness is assumed. Binary tournament selection, recombination, and mutation operators are used to create a child population Q_0 of size *N*. From the first generation onward, the procedure is different. The elitism procedure for $t \geq 1$ and for a particular generation is shown in the following:

$R_t = P_t \cup Q_t$	combine parent and children population
F = fast-non-dominated-sort R_t	$F = (F_1, F_2, ...)$,all non-dominated front of R_t
until $\mid P_{t+1} \mid + \mid F_i \mid < N$	till the parent population is filled
$P_{t+1} = \phi \quad and \quad i = 1$	
crowding-distance assignment (F_i)	calculate crowding distance in Fi
$P_{t+1} = P_{t+1} \cup F_i$	include i -th non-dominated front in the parent

```
                                          pop
$i = i + 1$                               check the next front for inclusion
Sort $(F_i, \prec_n)$                     sort in descending order using $\prec_n$
$P_{t+1} = P_{t+1} \cup F_i[1:(N-|P_{t+1}|)]$   choose the first $(N-|P_{t+1}|)$ element of cal $F_i$
$Q_{t+1} = make-new-pop(P_{t+1})$         use selection, crossover and mutation to create
$t = t + 1$                               a new population $Q_{t+1}$
```

First, a combined population $R_t = P_t \cup Q_t$ is formed. The population R_t will be of size *2N*. Then, the population R_t is sorted according to non-domination. The new parent population is P_{t+1} formed by adding solutions from the first front till the size exceeds *N*. Thereafter, the solutions of the last accepted front are sorted according to $\prec_n$ and the first *N* points are picked. This is how we construct the population P_{t+1} of size *N*. This population of size *N* is now used for selection, crossover and mutation to create a new population Q_{t+1} of size *N*. The NSGA-II procedure is shown in Fig. 4.7.It is important to note that we use a binary tournament selection operator but the selection criterion is now based on the niche comparison operator $\prec_n$.

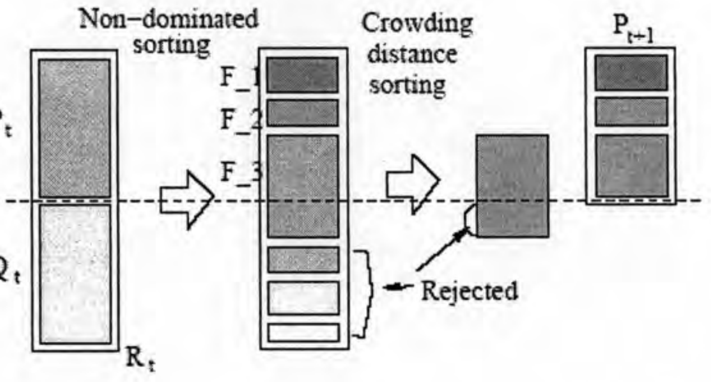

Figure 4.7: a procedure of NSGA-II.

Let us now look at the complexity of one iteration of the entire algorithm. The basic operations being performed and the worst case complexities associated with are as follows:

1. Non-dominated sort is O(mN^2),
2. Crowding distance assignment is O(*mN log N*), and
3. Sort on $\prec_n$ is O((*2N log 2N*).

As can be seen, the overall complexity of the above algorithm is O(mN^2) .

The diversity among non-dominated solutions is introduced by using the crowding comparison procedure which is used in the tournament selection and during the population reduction phase. Since solutions compete with their crowding distance (a measure of density of solutions in the neighborhood), no extra niching parameter (such as σ_{share} needed in the NSGA) 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.

It is interesting to note here the connection of this algorithm with the algorithm proposed by Rudolph. Since the non-dominated front finding algorithm used in Rudolph's algorithm is $O(mN^2)$ for each front, Rudolph control's the complexity of his algorithm by working with just the first few fronts in the parent and the child populations and treating the rest of the individuals in the child population at par. With the availability of a fast non-domination sorting algorithm we can now afford to combine the parent and child populations and do a complete sort to identify all the fronts and allocate fitness accordingly.

4.4.5 Strength Pareto Evolutionary Algorithm (SPEA): a brief overview

Zitzler and Thiele (1998) proposed an elitist evolutionary algorithm, which they called the strength Pareto EA (SPEA). This algorithm introduces elitism by explicitly maintaining an external population $\overline{P}$.This population stores a fixed number of the non-dominated solutions that are found until the beginning of a simulation. At every generation, newly found non-dominated solutions are compared with the existing external population and the resulting non-dominated solutions are preserved. The SPEA does more than just preserving the elites; it also uses these elites to participate in the genetic operations along with the current population in the hope of influencing the population to steer towards good regions in the search space.

This algorithm begins with a randomly created population P_0 of size N and an empty external population $\overline{P}_0$ with maximum capacity of $\overline{N}$. In any generation t, the best non-dominated solutions (belonging to the first non-dominated front) of the population P_t are copied to the external population $\overline{P}_t$. Thereafter, the dominated solution in the modified external population are found and deleted from the external population. In this way, previously found elites which are now dominated by new elite solution get deleted from the external population. What remains in the external population are the best non-dominated solutions of a combined population containing old and new elites. If this process is continued over many generations, there is danger of the external population being overcrowded with non-dominated solutions. In order to restrict the population to over-grow, the size of external population is less than $\overline{N}$, all elites are kept in the population. However, when the size exceeds $\overline{N}$, not all elites can be accommodated in the external population. This is where the investigator of the SPEA considered satisfying the second goal of multi-objective optimization. Elites which are less crowed in the non-dominated front are kept. Only that many solutions which are needed to maintain the fixed population size $\overline{N}$ are retained. The investigators suggested a clustering method to achieve this task. We shall describe this clustering method a little later.

Once the new elites are preserved for the next generation, the algorithm then turns to the current population and uses genetic operators to find a new population. The first step is to assign fitness to each solution in the population. It was mentioned earlier that the SPEA also uses the external population $\overline{P}_t$ in its genetic operations. In addition to the assigning of fitness to the current population members, fitness is also assigned to external population members. In fact, the SPEA assigns a fitness (called the strength) S_i to each member i of the external population first. The strength S_i is proportional to the number (n_i) of current population members that an external solution i dominates:

$$S_i = \frac{n_i}{N+1} \qquad (4.8)$$

In other words, the above equation assigns more strength to elite which dominates more solutions in the current population. Division by (N+1) ensures that the maximum value f the strength of any external population member is never one or more. In addition, a non-dominated solution dominated solution dominating a fewer solutions has a smaller (or better) fitness.

Thereafter, the fitness of a current population member j is assigned as one more than the sum of the strength values of all external population members which weakly dominate j:

$$F_j = 1 + \sum_{i \in p_t \wedge i \prec j} S_i \qquad (4.9)$$

The addition of one makes the fitness of any current population member P_t to be more than the fitness of any external population member $\overline{P}_t$. This method of fitness assignment suggests that a solution is smaller fitness is better. With these fitness values, a binary tournament selection procedure is applied to combined $(\overline{P}_t \cup P_t)$ population to choose solutions with smaller fitness values. Thus, it likely those external elites will be emphasizing during this tournament procedure. As usual, crossover and mutation operators are applied to the mating pool and a new population P_{t+1} of size N is created.

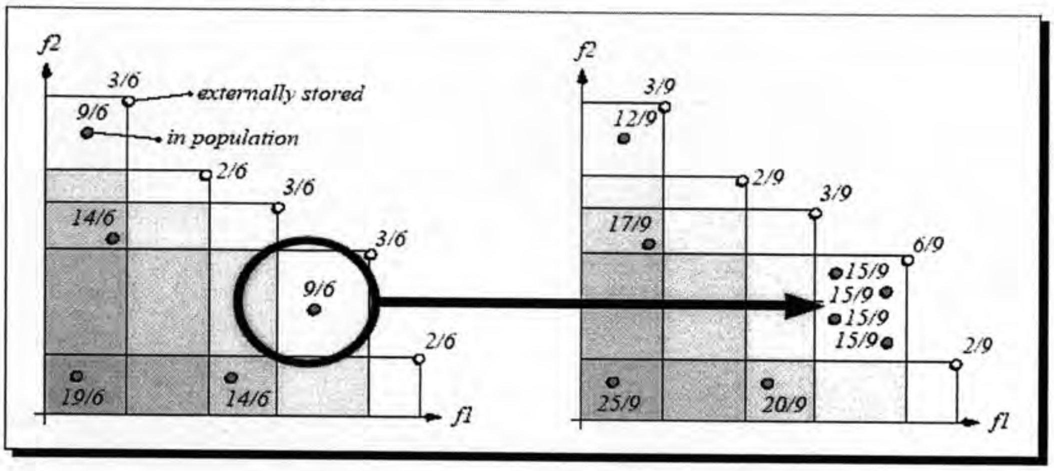

Figure 4.8: Demonstration of the niching mechanism used in SPEA. By adding three individuals to a particular niche, the reproduction probabilities of the niche members are decreased (i.e., the fitness values are increased).

Zitzler and Thiele will describe one iteration of the algorithm in step-by-step format. Initially, $\overline{P}_t = \phi$ is assumed.

Strength Pareto Evolutionary Algorithm (SPEA)

Step 1 Find the best non-dominated set $F_1(P_t)$ of P_t. Copy these solutions to $\overline{P}_t$, or perform $\overline{P}_t = \overline{P}_t \cup F_1(P_t)$.

Step 2 Find the best non-dominated solution $F_1(P_t)$ of the modified population $\overline{P}_t$ and delete all dominated solutions, or perform $\overline{P}_t = F_1(\overline{P}_t)$.

Step 3 If $|\overline{P}_t| > N$, use a clustering technique to reduce the size to $\overline{N}$. Otherwise, keep $\overline{P}_t$ unchanged. The resulting population is the external population $\overline{P}_{t+1}$ of the next generation.

Step 4 Assign fitness to each elite solution $i \in \overline{P}_{t+1}$ by using equation (4.8). Then, assign fitness to each population member $j \in P_t$ by using equation (4.9).

Step 5 Apply a binary tournament selection with these fitness values (in a minimization sense), a crossover and mutation operator to create the new population P_{t+1} of size N from the combined population $(\overline{P}_{t+1} \cup P_t)$ of size $(\overline{N} + N)$.

Step 3 and 5 result in the new external and current population, which are then processed in the next generation. This algorithm continues until a stopping criterion is satisfied.

4.4.5.1 Clustering Algorithm

The clustering algorithm which reduces the size of external population $\overline{P}_t$ of size $\overline{N}'$ to $\overline{N}$ where $(\overline{N}' > \overline{N})$.

At first, each solution in $\overline{P}_t$ is consider to reside in a separate cluster. Thus, initially there are $\overline{N}'$ clusters. Thereafter, the cluster-distance between all pairs of clusters are calculated. In general, the distance d_{12} between two clusters C_1 and C_2 is defined as the average Eucilidean distance of all pairs of solutions ($i \in C_1$ and $j \in C_2$), or mathematically:

$$d_{12} = \frac{1}{|C_1||C_2|} \sum_{i \in C_1, j \in C_2} d(i,j). \qquad (4.10)$$

The distance $d(i,j)$ **can be computed in the decision variable space or in the objective space. The proposers of the SPEA preferred to use the latter. Once all** $\binom{\overline{N}'}{2}$ cluster- distances are computed, the two clusters with the minimum cluster-distance are combined together to form one bigger cluster. Thereafter, in each cluster, the solution with the minimum average distance from other solutions in the cluster is retained and all other solutions are deleted. The following is the algorithm in a step-by-step format.

Clustering Algorithm

Step C1 Initially, each solution belongs to a distinct cluster or $C_i = \{i\}$, so that $C = \{C_1, C_2, \ldots, C_{\overline{N}'}\}$.

Step C2 If $|C| \leq \overline{N}$, go to Step C5. Otherwise, go to Step C3.

Step C3 For each pair of clusters (there are $\binom{\overline{C}}{2}$ of them), calculate the cluster-distance by using equation (4.9). Find the pair (i_1, i_2) which corresponds to the minimum cluster-distance.

Step C4 Merge the two clusters C_{i1} and C_{i2} together. This reduces the size of C by one. Go to step C2.

Step C5 Choose only one solution from each cluster and remove the others from the clusters. The solution having the minimum average distance from other solutions in the cluster can be the representative solution of a cluster.

Since in Step C5, all but one representative solution is kept, the resulting set has at most $\overline{N}$ solution.

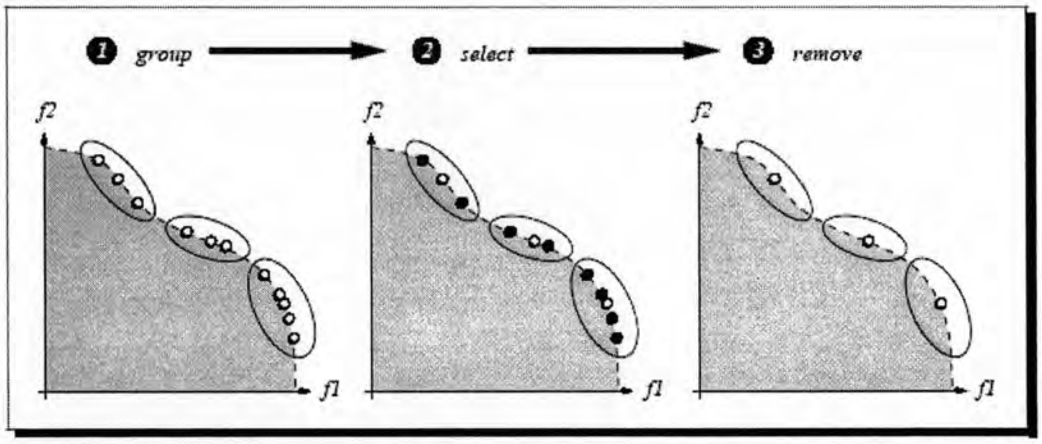

Figure 4.9:The principles of pruning a non-dominated set: i) solutions close to each other are grouped into clusters, ii) per cluster a representative solution is determined, and iii) the remaining solutions are removed from the set. Note that here clustering is performed in objective space, although distance may also be defined in decision space or individual space.