Part II

Computational Intelligence Algorithms

Chapter 5

Population-based Single-objective Algorithms

"One bee makes no swarm." – French proverb

This chapter provides an overview of two CI algorithms that are required as background for the DVEPSO algorithm that is introduced later in the thesis, as well as the algorithms against which DVEPSO are compared to (refer to Chapters 6, 7, 9 and 11). Section 5.1 discusses PSO, while GAs are discussed in Section 5.2.

5.1 Particle Swarm Optimisation

This section discusses the particle swarm optimisation (PSO) algorithm and the various steps of the algorithm. Section 5.1.1 discusses how the swarm is initialised and the conditions that will cause the algorithm to stop running are discussed in Section 5.1.2. The calculation of a particle's velocity is discussed in Section 5.1.3 and Section 5.1.4 discusses the calculation of a particle's position. Section 5.1.5 discusses the calculation of a particle's personal best and the swarm's global best.

Inspired by the social behaviour of bird flocks, Eberhart and Kennedy [94] introduced PSO. The PSO algorithm maintains a swarm of particles, where each particle represents a solution of the optimisation problem under consideration. Each particle moves through the search space and the particle's position in the search space is updated based on its own experience (cognitive information), as well as the experience of its neighbours (social information). The particle's position that produced the best solution so far is referred to as its personal best or *pbest*. The position that lead to the best overall solution by all particles in a pre-defined neighbourhood, i.e. either the best of the neighbourhood's particles' pbests or the best of the current positions of the neighbourhood's particles, is called the neighbourhood best or *nbest*.

The first PSOs introduced by Eberhart and Kennedy are the global best PSO, or *gbest* PSO, and the local best PSO, or *lbest* PSO. The gbest PSO defines the neighbourhood of each particle as the whole swarm. In this case the neighbourhood best is also referred to as the global best or *gbest*.

The PSO algorithm is described in Algorithm 1. The main steps of the algorithm are described in more detail below.

Algorithm 1 PSO Algorithm

```
1. create and initialise a swarm
2.     while stopping condition has not been reached
3.         for each particle in swarm do
4.             set pbest using Equation (5.6)
5.         set nbest using Equation (5.7) or Equation (5.8)
6.         for each particle in swarm do
7.             calculate new velocity using Equation (5.2) or Equation (5.4)
8.             calculate new position using Equation (5.5)
```

Before the PSO algorithm can run, certain values have to be set during the intialisation of the algorithm. The next section discusses the initialisation of the PSO.

5.1.1 Initialising the Swarm

The first step of the PSO algorithm initialises each particle's initial position, velocity and *pbest*, and sets swarm size, neighbourhood size, and the control parameters.

When initialising the particles' positions, it should be done in such a way that the particles uniformly cover the search space [55]. Therefore, assuming that the optimum is located within the domain defined by the vectors $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ (the minimum and maximum range of the decision variables in each dimension), an efficient initialisation method for the particles' positions is [55]:

$$x_j(0) = x_{min,j} + r_j(x_{max,j} - x_{min,j}), \quad \forall j = 1, 2, \ldots, n_x \tag{5.1}$$

where $r_j \sim U(0, 1)$, n_x refers to the dimension of the decision vector $\mathbf{x}$ and x_j is the j-th dimension of $\mathbf{x}$. The random value of r_j should be generated with an uniform number generator to ensure an uniform spread of solutions after initialisation.

The particles obtain both random positions and random moving directions if their positions are randomly initialised (as indicated in Equation (5.1)) [55]. Therefore, the initial velocities are normally set to zero. However, if the velocities are randomly initialised, the velocity values should be chosen carefully, since their values can lead to large position updates causing the particles to move outside the search space within the first few iterations of the run.

The *pbest* of each particle is set to its initial position, i.e. $\mathbf{y}_i(0) = \mathbf{x}_i(0)$. The PSO control parameters are set to values that lead to convergent behaviour [63], for example inertia weight, $w = 0.72$ and $c_1 = c_2 = 1.49$ (refer to Equations (5.2) and (5.4)). However, it should be noted that optimal values for the control parameters that lead to convergent behaviour are problem dependent.

The next section discusses conditions that are used to determine when a PSO algorithm stops running.

5.1.2 Stopping conditions

The PSO algorithm will continue to execute until a specific stopping condition has been reached. The most common stopping conditions used are:

- Running a fixed number of iterations (or function evaluations).
- Stopping when an acceptable solution has been found. If the true optima is known, then for SOOPs a stopping condition can be to stop if the error between the found optima and the true optima is smaller than a specified value. However, for MOOPs and for SOOPs, if the true optima is unknown, a stopping condition can be to stop if the value of a specific performance measure is better than a specified threshold.

5.1.3 Calculating the Velocity

This section provides information about the general method that is used to calculate the velocity of each particle, calculating a particle's velocity using an inertia weight, and clamping the velocities of particles to reduce the step size of the particles.

General Calculation of Velocity

This section discusses the calculation of the particles' velocities. First a general calculation of the velocities are discussed and then modifications to the general calculation of velocity are discussed.

The velocity of a particle is calculated as follows:

$$\mathbf{v}_i(t+1) = \mathbf{v}_i(t) + c_1\mathbf{r_1}(t)[\mathbf{y}_i(t) - \mathbf{x}_i(t)] + c_2\mathbf{r_2}(t)[\hat{\mathbf{y}}_\mathbf{N}(t) - \mathbf{x}_i(t)] \tag{5.2}$$

where $\mathbf{v}_i(t)$ and $\mathbf{x}_i(t)$ are the velocity and position of particle i at time step t respectively; $\hat{\mathbf{y}}_\mathbf{N}(t)$ represents the *nbest* of neighbourhood N (calculated using Equation (5.7) or Equation (5.8)) and $\mathbf{y}_i(t)$ represents the *pbest* (calculated using Equation (5.6)) at time t; $c_1\mathbf{r_1}(t)[\mathbf{y}(t) - \mathbf{x}(t)]$ is the cognitive component of the velocity and $c_2\mathbf{r_2}(t)[\hat{\mathbf{y}}_\mathbf{N}(t) - \mathbf{x}(t)]$ is the social component of the velocity; c_1 and c_2 are positive acceleration coefficients that influence the contributions of the cognitive and social components respectively; and $\mathbf{r_1}, \mathbf{r_2} \sim U(0,1)^{n_x}$ are random values sampled from an uniform distribution with n_x representing the number of decision variables or the dimension of the search space.

One problem with the general calculation of velocity using Equation (5.2) is that the velocity value of a particle can quickly become very large, especially when the particle is exploring an area in the search space that is far away from the particle's pbest or the swarms global best. A large velocity value results in a large position update, which results in the particle moving outside the feasible space. To overcome this problem, various modifications to the basic PSO have been suggested. Two of the modifications that have been made to the general calculation of velocity are discussed in more detail below, namely clamping the velocities of the particles and using inertia weight in the calculation of velocity.

Velocity Clamping

Without any intervention, a particle's velocity may increase in such a way that it may move outside the boundary constraints of the problem. One way of managing this is to clamp the particle's velocity, i.e. if a particle's velocity exceeds a predefined maximum velocity value, the particle's velocity is set to the maximum velocity value [51]. Mathematically, this is described as follows:

$$\mathbf{v}_i(t+1) = \begin{cases} \mathbf{v}_i'(t+1) & \text{if } \mathbf{v}_i'(t+1) < \mathbf{v_{max}} \\ \mathbf{v_{max}}(t+1) & \text{if } |\mathbf{v}_i'(t+1)| \geq \mathbf{v_{max}} \end{cases} \tag{5.3}$$

where $\mathbf{v_{max}}$ is the maximum velocity in each dimension and $\mathbf{v}_i'$ is the velocity of particle i, calculated using Equation (5.4).

Velocity clamping does not prevent a particle from moving outside the boundaries of the feasible space. However, it does restrict the step sizes of the particles. It should be noted that the selection of a $\mathbf{v_{max}}$ value should be carefully chosen and is problem dependent. A large $\mathbf{v_{max}}$ value will increase the swarm's global exploration, but the larger step sizes of the particles may cause the particles to jump over good solutions to continue searching in an area of the search space that does not contain good solutions [55]. A small $\mathbf{v_{max}}$ value will increase the swarm's local exploitation, but a too small $\mathbf{v_{max}}$ value may cause the swarm to only explore local good regions and not other good regions that are further away. Furthermore, with a too small $\mathbf{v_{max}}$ value the swarm may become trapped in local optima [55].

Another approach that is followed to prevent large position updates of particles, is using an inertia weight. The next section discusses how particles' velocities are calculated using an inertia weight and the effect that the inertia weight has on the velocities of the particles.

Calculating the Velocity using Inertia Weight

Shi and Eberhart [136] introduced the concept of an inertia weight to control the influence of previous flight magnitude (step size and direction) on the new velocity, i.e. the momemtum of a particle. The velocity calculation of Equation (5.2) can therefore be adapted as follows:

$$\mathbf{v}_i(t+1) = w\mathbf{v}_i(t) + c_1\mathbf{r_1}(t)[\mathbf{y}_i(t) - \mathbf{x}_i(t)] + c_2\mathbf{r_2}(t)[\hat{\mathbf{y}}_\mathbf{N}(t) - \mathbf{x}_i(t)] \tag{5.4}$$

where w represents the inertia weight.

The value of w influences the exploration and exploitation ability of the swarm [136, 63]. Shi and Eberhart [136] investigated the effect of w values in the range $[0, 1.4]$ and the study's results indicated that better convergence was achieved with $0.8 \leq w \leq 1.2$, and $w > 1.2$ resulted in more failures in finding the global optimum due to particles leaving the search space.

Let $c_1 = c_2 = 0$ in Equation (5.4). Then, if $w > 1$, the particles' velocities will keep increasing over time, or will keep increasing over time until the maximum velocity is reached if velocity clamping is used. This increase in the velocity will cause the swarm to diverge and therefore, large w values facilitate more exploration. On the other hand, if $w < 1$, the velocities will keep decreasing over time until they reach zero or values close to zero. Therefore, small w values lead to local exploitation, but too small values reduce the swarm's exploration ability.

However, when $c_1, c_2 \neq 0$, the effect of w is not that easy to predict. According to Shi and Eberhart [136] w values close to 1.0 seems preferable. However, according to a study by Van den Bergh and Engelbrecht [152], to ensure convergence, w should be chosen in such a way that it adheres to the following relation: $w > \frac{1}{2}(c_1 + c_2) - 1$. From this relation it is clear that the values of the control parameters w, c_1 and c_2 cannot be selected independently to ensure convergence. Furthermore, the best values for these control parameters are problem dependent.

5.1.4 Calculating the Position

Once the new velocity of a particle has been calculated, its new position can be determined by adding the velocity to its current position as follows:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i + \mathbf{v}_i(t+1) \tag{5.5}$$

When a particle moves outside the boundaries of the search space, the particle's position is calculated differently from Equation (5.5) to pull the particle back into the search space. Various ways exist to manage boundary constraint violations, as discussed below.

Dealing with Boundary Constraints

Most optimisation problems have boundary constraints (refer to Section 2.1) and therefore a particle should be prevented from moving outside the search space of the problem. If the solution is in close proximity of the bounds, it may be beneficiary to allow a particle to move outside the bounds to enable exploration in the proximity of the optima in the hope that the particle may find the optima. However, once a particle has moved outside the bounds, it should not be allowed to become an attractor (the particle's position should not be selected as either a *pbest* or *nbest*) to ensure that the particle doesn't attract other particles to the area outside the search space.

According to Chu *et al* [25], there are three basic approaches that are widely used to manage boundary constraint violations, namely:

- Random, where if any dimension of a particle's position is outside the search space, a random value from an uniform distribution between the lower and upper boundaries of the violating dimension is assigned to the violating dimension of the particle's position.
- Absorbing, where if a particle moves outside the search space, the dimension that is violating the bounds are set to the boundary of that dimension, so that it seems as though the particle has been absorbed by the boundary.
- Reflection, where if a particle moves outside the search space, the boundary acts like a mirror that reflects the projection of the particle's displacement by flipping the direction of the particle's velocity.

According to Engelbrecht [55], the following approaches can also be used to manage boundary constraint violations:

- Repairing, where if a particle moves outside the search space, it is pulled back into the search space by the *pbest* and *nbest*. For example, if a particle's position in dimension j violates the boundary constraints, the particle's velocity in dimension j, v_j, is set to zero to eliminate the influence of momentum for the j-th dimension, so that this dimension will be pulled back towards dimension j of *pbest* and *nbest*.

However, the particle will only be pulled back within the search space if the *pbest* is still within the search space.

- *pbest* selection, where particles are allowed to cross the boundaries, but if the position of a particle is not within the search space, its position cannot become the particle's *pbest*.

5.1.5 Calculating the *pbest* and *nbest*

This section discusses how the *pbest* and *nbest* is calculated. The equations that are used to update the *pbest* and *nbest* are provided. This section discusses issues that influence the manner in which the *pbest* and *nbest* are updated, namely whether the updates are done in a synchronous or asynchronous way and how the particles in the swarm are connected to each other.

For minimisation problems, the *pbest* at time $t+1$ is calculated as [94]:

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i(t)) \end{cases} \tag{5.6}$$

where n_x represents the number of decision variables and the fitness function is represented by $f : \mathbb{R}^{n_x} \rightarrow \mathbb{R}$.

The *nbest* at time t can be calculated as [55]:

- The best *pbest* found so far by all particles in the neighbourhood N_j, calculated as:

$$\hat{\mathbf{y}}_{N_j}(t) \in \{N_j \mid f(\hat{\mathbf{y}}_{N_j}(t)) = \min\{f(\mathbf{y}_i(t))\},\ \forall \mathbf{y}_i \in N_j\} \tag{5.7}$$

- The best position of all positions found by the particles in neighbourhood j at a specific time step t, calculated as:

$$\hat{\mathbf{y}}_{N_j}(t) \in \{N_j \mid f(\hat{\mathbf{y}}_{N_j}(t)) = \min\{f(\mathbf{x}_i(t))\},\ \forall \mathbf{x}_i \in N_j\} \tag{5.8}$$

Synchronous and Asynchrounous Updates

The *pbest* and *nbest* values can be updated in either a synchronous or asynchronous manner [21]. *Synchronous* updates of these values are done when all the particles' positions are first calculated and their *pbest* values are updated, and then the *nbest* value is calculated. Algorithm 1 uses synchronous updates. With synchronous updates, feedback about the best search region is only given once per iteration and all of the particles' knowledge about the *nbest* is the same and updated at the same time. Therefore, Carlisle and Dozier [21] suggest that synchronous updates is suitable for the *gbest* PSO. With *asynchronous* updates, the new *nbest* position is calculated after each particle's position and *pbest* update. Immediate feedback about the best regions in the search space is provided and feedback occurs many times during one iteration. Therefore, according to Carlisle and Dozier [21], asynchronous updates are more suitable for the *lbest* PSO.

Neighbourhood Topologies of Particle Swarm Optimisation

The topology or connections between particles in a swarm influences the communication flow between the various particles. Various topologies have been developed for PSO, but only the topologies used by the *lbest* and *gbest* PSO are discussed in this section. The reader is referred to [26, 55, 95, 126] for more information on the different PSO topologies.

The **star** topology connects all particles to each other and therefore every particle can communicate with all other particles. In this case the neighbourhood of each particle is the whole swarm and therefore the *nbest* is also referred to as the global best or *gbest*. Equations (5.7) and (5.8) are then adapted to calculate the *gbest* as follows: The *gbest* at time t can be calculated as either [55]:

- The best *pbest* found so far by all particles in the swarm, calculated as:

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0(t), \dots, \mathbf{y}_{n_p}(t)\} | f(\hat{\mathbf{y}}(t)) = min\{f(\mathbf{y}_0(t)), \dots, f(\mathbf{y}_{n_p}(t))\} \tag{5.9}$$

  where n_p is the number of particles in the swarm and $\hat{\mathbf{y}}$ is the *gbest*.

- The best position of all positions found by the particles at a specific time step t, calculated as:

$$\hat{\mathbf{y}}(t) = min\{f(\mathbf{x}_0(t)), \dots, f(\mathbf{y}_{x_{n_p}}(t))\} \tag{5.10}$$

Since all particles are connected to each other with a star topology, information about the *gbest* can be quickly distributed to all particles, attracting all particles to the best part of the search space, which may lead to faster convergence. However, since a PSO with a star topology has only one attractor (the *gbest*), and the extent of coverage of the search space is less than with a less connected topology, if the global optima is not close to the *gbest*, the PSO may be trapped in local optima [95, 55]. The *gbest* PSO has a star topology.

A less connected topology than the star topology is the **ring** topology, where each particle communicates directly with only a predefined number of immediate neighbours. It is important to note that the different neighbourhoods in the swarm overlap so that a specific particle can belong to more than one neighbourhood. Allowing particles to belong to more than one neighbourhood enables communication or exchange of information between neighbourhoods. Information about the best solution flows slower through the ring topology than through the star topology, leading to slower convergence. However, with the ring topology a larger area of the search space is covered than with the star toplogy, and it has many attractors (*nbests*), and therefore it is less susceptible to local optima [95, 55]. The *lbest* PSO has a ring toplogy.

Many variations of the original PSO have been developed through the years. However, these variations are beyond the scope of this thesis and the reader is referred to [4, 5, 26, 52, 55, 126] for more information on the various PSO algorithms.

5.2 Genetic Algorithms

The concept of genetic algorithms (GAs) was first described by Holland [82]. GAs are based on concepts of Darwanian evolution. Every living organism consists of cells, where each cell contains the same set of *chromosomes*, i.e. an organised structure of DNA and protein. Each chromosome consists of blocks of DNA referred to as *genes*, where each gene encodes or represents a particular protein or trait, such as the organism's hair colour. The possible values that a gene can have are called *alleles*, for example the eye colour that can be blue, green or brown [121]. Using the metaphor of "survival of the fittest", a GA uses a population of individuals to evolve towards better solutions. When an optimisation problem is solved using a GA, each individual's characteristics are represented by a chromosome. Each chromosome is a combination of the decision variables that have to be optimised, where each decision variable is referred to as a gene.

Algorithm 2 provides a template for a GA.

Algorithm 2 Genetic Algorithm

```
1. create and initialise the population
2.   while stopping condition has not been reached
3.     for each individual of the population do
4.       evaluate the fitness
5.     select parents for cross-over
6.       perform cross-over on parents to produce offspring (children)
7.     select offspring for mutation
8.       perform mutation on offspring
9.     select a new population for the next generation
```

This section discusses GAs and the various steps of the GA. Section 5.2.1 discusses GA control parameters. Evaluation of the various solutions that are found is discussed in Section 5.2.2 and various selection operators are discussed in Section 5.2.3. Section 5.2.4 discusses the cross-over operator and Section 5.2.5 discusses the mutation operator.

5.2.1 Initialising the Population

The original GA as proposed by Holland used a binary representation for the chromosome. An optimisation problem with a n_x-dimensional search space is then represented by chromosomes that consist of n_x bit-valued strings, i.e. $x_j \in \{0, 1\}$, where each bit string represents a decision variable. If the decision variables are binary variables, the length of the chromosome is n_x bits. However, if the variables have nominal values, the chromosome has n_x bit vectors of length n_d, where each decision variable can have 2^{n_d} values. If the variables have continuous values, a mapping function is used to convert the continuous value to a bit vector. The mapping function is formulated as $\phi : \mathbb{R} \to \{0, 1\}^{n_d}$. It should be noted that GAs and operators have been developed where floating point presentation can be used directly [81].

As for PSO, chromosomes are initialised to uniformly cover the search space.

5.2.2 Fitness Evaluation

In nature, usually the fittest of the specie survives. Therefore, a GA should use the better solutions to create the new solutions and the best solutions should survive to the next generation. In order to ensure that the GA adheres to this principle, some method of quantifying the quality of a solution represented by a chromosome should be used. Similar to the PSO, this is done using a fitness function that maps the chromosome to a scalar value, expressing the quality of the candidate solution.

5.2.3 Selection Operator

Various approaches exist to select solutions from a specific population, called selection operators. When GAs are used to solve optimisation problems, solutions are selected for the following three steps in the algorithm:

- Solutions from the parent population are selected for cross-over to create new solutions called *offspring* (refer to steps 5 and 6 in Algorithm 2).
- A solution from the offspring population is selected for mutation to create new solutions or offspring (refer to steps 7 and 8 in Algorithm 2).
- Solutions are selected from the parents and offspring for the next generation of the GA (refer to step 9 in Algorithm 2).

Selection operators can be quantified according to their selection pressure (or take-over-time), i.e. the time it takes for the best solution to occupy all but one population slot (individual) if a specific selection operator is repeatedly applied to a population [69]. When a selection operator has a high selection pressure, the population will lose diversity quickly, which may lead to premature convergence to sub-optimal solutions. On the other hand, if the selection pressure is low, the GA's search procedure will behave more like a random search process [40], since the best solutions are not emphasised during selection.

The most common selection operators are tournament selection, proportionate selection, rank-based selection, random selection and elitist selection [40, 55].

Tournament Selection

Tournament selection randomly selects n_{ts} number of individuals from the population (n_{ts} is smaller than the total number of individuals in the population, n_s), compares their fitnesses against each other and then selects the individual with the best fitness. If tournament selection is performed without replacement, the selected individuals are not considered for the next selection. If selection is done for cross-over, tournament selection is performed repeatedly until the required number of parents for cross-over have been selected. If $n_{ts} = 2$, tournament selection is referred to as binary tournament selection.

If n_{ts} is not too large, tournament selection has a small selection pressure, since the best individual has a smaller chance of being selected. However, if n_{ts} is too large, it may occur that the best individual is selected more than once. Furthermore, if n_{ts} is too small, a bad individual may be selected. Therefore, the value of n_{ts} plays a huge roll in the selection pressure of tournament selection.

Proportionate Selection

With proportionate selection, the probability of selecting $\mathbf{x}_i$, $p(\mathbf{x}_i)$, is proportionate to the fitness of $\mathbf{x}_i$, calculated as [55]:

$$p(\mathbf{x}_i) = \frac{f(\mathbf{x}_i)}{\sum_{j=1}^{n_i} f(\mathbf{x}_j)} \tag{5.11}$$

with $p(\mathbf{x_i})$ the probability that individual $\mathbf{x}_i$ will be selected.

Proportionate selection has a large selection pressure and individuals with a good fitness value will have a better chance of being selected, which can decrease the diversity of the population in the next generation. Individuals are selected by sampling the distribution created using Equation (5.11). For example, individuals can be selected according to roulette wheel selection (RWS) where the fitness values are normalised by dividing the fitness value by the maximum fitness value. The probability distribution is then a roulette wheel, where the size of each slice is proportional to the normalised selection probability of an individual. Selecting an individual is then similar to spinning the roulette wheel, recording which slice ends up at the top and then selecting the corresponding individual. With this approach, the wheel has to be spun as many times as the number of individuals that have to be selected [40, 55]. In other words, random numbers have to be created (simulating a spin) for each individual that has to be selected. However, instead of creating N random numbers for N individuals that have to be selected, stochastic universal sampling (SUS) can be used where only one random number is selected for the entire selection process [40]. If N individuals have to be selected, a set of N equally spaced numbers is created:

$$R = \left\{ r,\ r + \frac{1}{N},\ r + \frac{2}{N}, \ldots,\ r + \frac{N-1}{N} \right\} \%1 \tag{5.12}$$

where % is the modulus operator.

The corresponding individuals, based on their selection probability, are then selected according to set R.

Rank-based Selection

Rank-based selection ranks the individuals' fitness values and the individuals' selection probability is then calculated based on the rank and not based on the absolute fitness values. Since the selection is independent of the actual fitness value, the best individual will not dominate the selection process. Therefore, rank-based selection has a small selection pressure. The ranking of the fitness values can be done in many ways, of which two are a linear ranking approach or an exponential ranking approach [54, 55].

Random Selection

Random selection selects an individual randomly from the population, and all individuals have an equal chance of being selected. Therefore, random selection has a small selection pressure, since weaker individuals have the same chance of being selected than the best individual.

Elitist Selection

The goal of elitist selection is to ensure that the best individuals of the current population survive to the next generation. The best individuals are added to the new population without performing mutation. Therefore, elitist selection has a large selection pressure.

However, if many individuals survive to the next generation, the population's diversity will decrease.

5.2.4 Cross-Over Operator

The goal of cross-over is to produce offspring that hopefully produce better solutions than their parents. A cross-over operator is applied to a specified number of selected parent solutions and their genetic material or genes are recombined to produce one or more offspring.

When the chromosomes are represented by bit vectors, i.e. the genes have binary representations, uniform- [143], one-point- [82, 92], two-point and n-point [92] cross-over can be used. If the genes have continuous values, other cross-over methods are used, for example linear-, blend- and simulated binary crossover [40]. More information on the various cross-over approaches for binary-values genes and continuous-valued genes can be found in [40, 139, 140].

In order to preserve some good individuals, the cross-over operator is not applied to all parents, but only to a percentage of the parent population. The percentage of the populuation that is selected for cross-over is specified by the cross-over probability, p_c. If p_c is small, only a few new solutions (offspring created with cross-over) are introduced to the selection pool, leading to a smaller area of the search space being searched. However, if p_c is high, many new solutions are created and only a few of the individuals are preserved in the next generation. This will lead to a bigger area of the search space being searched, but good genetic material may get lost in the process.

5.2.5 Mutation Operator

When cross-over is applied, good genetic material may get lost in the process. This problem can be addressed through mutation, since mutation introduces more diversity into the population. When the mutation operator is applied to the selected offspring, the value of randomly selected elements of the chromosome is changed. Each chromosome has a mutation probability p_m of being selected for mutation. If p_m is small, only a small number of genes are mutated, thereby increasing the diversity of the solutions, but not changing the individuals too drastically. Furthermore, a small p_m may re-introduce genetic material that got lost during cross-over. However, if p_m is large, many genes are changed, almost no genetic material of previous solutions are preserved, and the search process becomes almost similar to a random search [140].

Mutation changes the value of selected genes. In the case of bit-valued genes, the bit value is simply negated. In the case of floating-point values, a mutational step size is sampled from some zero-mean distribution and added to the gene's value [159].

5.3 Summary

This chapter discussed two computational intelligence algorithms, namely PSO and GA, that are required as background for the vector evaluated approaches discussed in Chapter 7, as well as the DMOO algorithms discussed in Chapters 9 and 11.

Section 5.1 provided information about PSO and the various steps of the basic PSO algorithm. The various steps of the basic PSO that were discussed are: the initialisation of the PSO swarm, the conditions under which the PSO algorithm will stop running, the calculation of a particle's position and velocity, and the calculation of a particle's personal best and the swarm's global best.

GAs and the various steps of a GA were discussed in Section 5.2. The various steps of a GA that were discussed, are: how to initialise the population, how to calculate the fitness of an individual, selection of individuals, and cross-over and mutation operators.

The next chapter discusses population-based approaches that were used to solve optimisation problems with more than one objective.

Chapter 6

Population-based Multi-objective Optimisation Algorithms

"When it is obvious that the goals cannot be reached, don't adjust the goals, adjust the action steps." – Confucius

Most problems in real-life have more than one goal or objective that are in conflict with one another - by improving one objective the solutions get worse with regards to the other objective(s). Therefore, the goal of a MOO algorithm is to find the set of optimal trade-off solutions called the POF.

This chapter discusses three MOO algorithms that are required as background for Chapter 8. Chapter 8 discusses versions of these algorithms adapted for DMOO. The performance of the adapted versions of these algorithms are compared against the DMOO algorithm presented in this thesis, namely DVEPSO (refer to Chapter 11).

Section 6.1 provides a short overview of the MOO field. NSGA-II, a multi-population GA, is discussed in Section 6.2. A multi-population PSO, MOPSO is discussed in Section 6.3. Section 6.4 discusses a multi-population cooperative and competitive EA, CCEA. A summary of the chapter is provided in Section 6.5.

6.1 History of Multi-Objective Optimisation

Various CI algorithms have been used to solve MOOPs. This section provides a short summary of the major contributions in the field. The information provided in this section is by no means complete and the reader is referred to [35, 36, 38] for a more detailed discussion of MOO.

David Schaffer's VEGA approach [131] is generally seen as the first implementation of a multi-objective evolutionary algorithm (MOEA) [31]. VEGA is a multi-population approach where each sub-population solves only the one objective assigned to the sub-population. Proportional selection is performed on the sub-population based on only the one objective. All of the selected individuals are then placed together in a new combined population on which cross-over and mutation are applied using a standard GA. VEGA is discussed in more detail in Section 7.1.

However, after Schaffer presented VEGA, researchers mostly used aggregation of objective functions and lexicographic ordering to solve MOOPs [31]. Aggregation methods combine the objective functions into a single objective, which is then used as the fitness function. The objectives can be combined in both a linear and non-linear way. If non-linear aggregation is used, the new objective function, f^*, is created as a weighted sum of the various objective functions, $f_1, f_2, \ldots, f_k$, as follows:

$$f^* = \sum_{j=1}^{k} w_j f_j \tag{6.1}$$

where w_j is the weight of objective function f_j, indicating the relative importance of the specific objective function.

The main problem when using linear aggregation is that the algorithm struggles to generate solutions when the POF is nonconvex [40]. Lexicographic ordering orders the objectives according to importance. The objective that is considered as the most important is then chosen and optimised without considering the other objectives. This process is repeated, optimising all objectives one by one, in the order of importance, until all objectives have been optimised.

The introduction of the concept of *Pareto ranking* by Goldberg [68] changed the way in which EAs are used to solve MOOPs. Pareto ranking is illustrated in Algorithm 3.

Algorithm 3 Pareto ranking

```
1.  set population $P_r$ equal to the whole population $P$
2.  set $i = 0$
3.    while $P_r$ is not empty
4.      select individuals from $P_r$ that are non-dominated
5.      assign rank $i$ to the selected individuals and remove them from $P_r$
6.      increment $i$
```

The individuals with rank 1 are similar to non-dominated solutions stored in an external archive.

Even though Goldberg did not implement Pareto ranking himself, most MOEAs that followed after his suggestion, implemented Pareto ranking [60, 83, 141]. The goal of Pareto ranking is to enable an algorithm to converge to the POF.

The early MOO algorithms also faced the problem of preventing the EA from converging to a single solution [31]. To overcome this problem, Goldberg and Richardson suggested the use of a niching technique [45, 70] to increase the diversity of the found non-dominated solutions. A commonly used niching technique is fitness sharing [30, 70], where the fitness of an individual $\mathbf{x}_i$ is degraded by the presence of individuals that are within a specified niche radius (σ_{share}) of $\mathbf{x}_i$.

The introduction of the strength Pareto evolutionary algorithm (SPEA) by Zitzler [172] changed the type of MOEAs that were presented and are still developed today to solve MOO. SPEA incorporates elistim through the use of an archive that stores the non-dominated solutions that have been found so far by the algorithm. The archive is used to ensure that the non-dominated solutions reported at the end of the algorithm run are solutions that are non-dominated with regards to all solutions that have been found by the algorithm throughout the run. It should be noted that elitist selection with regards to GAs refers to ensuring that the best individuals of the current population survive to the next generation (refer to Section 5.2.3). However, in the context of MOO, elitism can be seen as retaining the best solutions obtained by the algorithm throughout the entire run, i.e. solutions that remained non-dominated throughout the entire run of the algorithm.

6.2 Non-dominated Sorting Genetic Algorithm II

NSGA-II, introduced by Deb *et al.* [47, 42], is an improved version of non-dominated sorting genetic algorithm (NSGA) developed by Srinivas and Deb [141]. This section discusses the various steps of NSGA-II. Section 6.2.1 discusses the basic NSGA-II. The fast non-dominated sorting approach used by NSGA-II is discussed in Section 6.2.3. Section 6.2.4 discusses the approach that is followed to maintain the diversity of the non-dominated solutions.

6.2.1 NSGA-II

NSGA-II is a multi-objective genetic algorithm (MOGA) that uses non-domination. The various steps of the algorithm is illustrated in Figure 6.1 and listed in Algorithm 4.

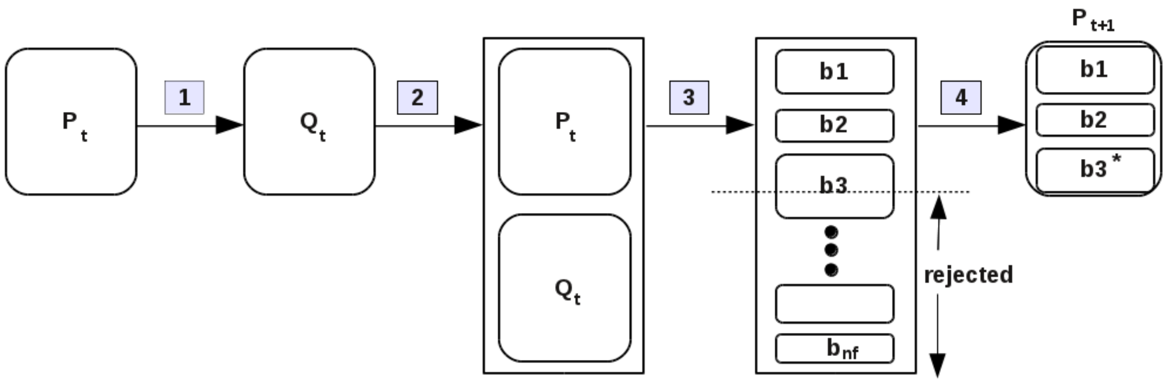


Figure 6.1: Steps of NSGA-II

Algorithm 4 NSGA-II

```
1.  create and initialise a random parent population
2.      while stopping condition has not been reached
3.          produce offspring (step 1 in Figure 6.1)
5.          combine parents and offspring in a combined population $R$ (step 2 in Figure 6.1)
6.          sort $R$ according to Pareto-ranking (step 3 in Figure 6.1)
7.          select individuals for the new population (step 4 in Figure 6.1)
```

The steps of NSGA-II highlighted in Figure 6.1 are discussed in more detail below.

6.2.2 Producing Offspring

The first step in Figure 6.1 is to produce offspring. Before the algorithm starts, an initial parent population, P_0, is randomly created. P_0 is then sorted based on non-domination, where each individual is assigned a fitness based on the individual's non-domination level. Binary tournament selection is performed to select parents for cross-over to produce n_p offspring. Once the offspring has been created, mutation is performed on the offspring.

6.2.3 Fast Non-dominated Sorting

After the offspring has been created, the parents and offspring are combined to create a new population, R, of size $2n_p$. This is the second step in Figure 6.1. Pareto-ranking is performed on R, i.e. R is sorted according to the individuals' level of non-domination. This sorting process is the third step in Figure 6.1. In order to speed up the sorting process, Deb *et al.* introduced the fast non-dominated search procedure for NSGA-II [47, 42].

Fast non-dominated search places the first individual of R in a new population P'. Each individual, $\mathbf{x}_i \in R$, is then compared against all individuals in P'. If $\mathbf{x}_i$ dominates any solutions in P', the dominated individuals are removed from P'. If $\mathbf{x}_i$ is dominated by any individual in R, $\mathbf{x}_i$ is ignored and not placed in P'. However, if $\mathbf{x}_i$ is non-dominated with regards to all individuals in R, $\mathbf{x}_i$ is placed in R. After all $\mathbf{x}_i \in R$ have been compared against the individuals in R, the individuals in P' is part of the first front. To determine the second front, all members of the first front are removed from R and not considered. The search process is then repeated. This whole process is repeated until all fronts have been found.

6.2.4 Selecting a New Population

Once all the individuals in R has been assigned a front, a new population, P_{t+1}, is selected for the next generation. This selection process is explained in Algorithm 5.

Algorithm 5 NSGA-II Population Selection

```
1.  if $|P_{t+1}| < n_p$
2.      for each front $b_i$
3.          if $|b_i| < (n_p - |P_{t+1}|)$
5.              add all individuals in $b_i$ to $P_{t+1}$
6.          else
7.              perform crowded sorting on $b_i$
8.              while $|P_{t+1}| < n_p$
9.                  add best individual from sorted $b_i$
10.                 remove best individual from sorted $b_i$
```

Crowded sorting [47, 42] is used to ensure diversity of solutions. In order to perform crowded sorting, the individuals' crowding distances are determined. The population is sorted with regards to each objective function in ascending order. Then, for each objective function:

- The boundary individuals, i.e the individuals with the highest and lowest objective function value, are assigned an infinite crowding distance value.
- The other individuals are assigned a crowding distance equal to the distance in objective space between the two adjacent points of the individual, i.e. the adjacent points on each side of the individual in objective space.

Each individual's crowding distance is then calculated as the sum of the individual's crowding distance values for each objective.

After the crowding distances have been calculated, crowded sorting are performed on the individuals. When two individuals are compared using the crowded sorting operator, then

- if two individuals have different non-domination ranks, the individual with the lowest domination rank is selected.
- if both individuals have the same non-domination ranks and therefore belong to the same front, the individual with the largest crowded distance is selected. This ensures that the individual that is located in the less dense area is selected and thereby increases the diversity of the set of non-dominated solutions.

In NSGA-II, NSGA's fitness sharing scheme is replaced with the crowded sorting operator. Another difference between the two algorithms is NSGA-II's faster non-dominating sorting procedure [42, 47]. These changes improves NSGA-II's performance with regards to computational complexity. Furthermore, the performance of NSGA-II is so good, that it has become a benchmark against which other MOOs algorithms are compared against [32].

It is important to note that NSGA-II does not make use of an archive, but preserves elitism through its selection mechanism. NSGA-II uses $(\mu + \lambda)$-selection [9, 134] where μ represents the number of parents and λ the number of offspring produced from the parents. In other words, NSGA-II selects the best μ individuals from both the parents and the offspring for the next generation.

A disadvantage of the crowded sorting operator is that it may cause a Pareto-optimal solution that is in a crowded area (with other non-dominated solutions that are not necessarily Pareto-optimal) to be deleted and non-dominated solutions that are not Pareto-optimal but located in a less crowded area to be preserved [38].

6.3 Multi-objective Particle Swarm Optimisation

The MOPSO algorithm was introduced by Coello Coello and Salazar Lechuga [33] as one of the first PSO algorithms extended for MOO. Algorithm 6 lists the various steps of the MOPSO algorithm.

Before the MOPSO algorithm can be executed, the swarm is initialised as discussed in Section 6.3.1. Section 6.3.2 discusses the calculation of the particles' velocity. The approach used to calculate the swarm's local guide is discussed in Section 6.3.3.

6.3.1 Initialising the Swarm

Similar to PSO, the first step of the MOPSO algorithm initialises each particle's initial position, velocity and *pbest*, and sets swarm size, neighbourhood size, and the control parameters. The particles' initial positions are initialised in such a way that the particles are uniformly spread over the search space. The particles' velocities are initialised to zero and their *pbests* are set to their current positions.

Algorithm 6 MOPSO Algorithm

```
1.  create and initialise a swarm
2.    while stopping condition has not been reached
3.      for each particle in swarm do
4.        calculate new velocity using Equation (5.4)
5.        calculate new position using Equation (5.5)
6.        manage boundary constraint violations
7.      update archive
8.      update the particles' allocation to hypercubes
8.      for each particle in swarm do
9.      update pbest
```

In addition to the PSO initialisation, the particles are evaluated and the positions of the particles that are non-dominated are stored in the archive. Furthermore, the search space that has been explored so far is divided into hypercubes and all particles are placed in a hypercube based on the particle's position in objective space.

6.3.2 Calculation of Velocity

MOPSO uses the velocity equation of PSO (refer to Equation (5.4)) to update the velocity of the particles. However, the *gbest* in Equation (5.4) is a global guide that MOPSO selects from the archive as follows:

- hypercubes containing more than one particle is assigned a fitness equal to $\frac{f(\mathbf{x})}{n_p}$ where $f(\mathbf{x}) > 1$ and n_p is the number of particles in the swarm. This ensures that hypercubes that contain more particles will have a lower fitness value.
- roulette-wheel selection is used on the fitness values to select the hypercube from which the guide is selected.
- a particle is randomly selected from the winning hypercube and selected as the global guide.

6.3.3 Calculation of *pbest*

MOPSO [33] updates each particle's *pbest* (referred to as the local guide) as follows: the new position of the particle is compared to the particle's *pbest*, taking all objective functions into account, and

- if the new position dominates the current *pbest*, the new position is selected as the *pbest*, otherwise
- if the new position is non-dominated with regards to the current *pbest*, the new *pbest* is randomly selected between the particle's position and the current *pbest*.

In contrast to NSGA-II, MOPSO uses an archive to preserve elitism. However, the original version of MOPSO struggled to converge to the true POF in the presence of many local POFs [34]. To overcome this problem, Coello *et al.* [34] introduced an updated version of MOPSO that uses a mutation operator. Initially, the mutation operator is applied to all particles, but then the number of particles being mutated decreases rapidly as the number of iterations increases. The goal of the mutation operator is to increase the swarm's exploration ability. However, the mutation operator is not only applied to the particles, but also to the range of each decision variable of the MOOP. This leads to the whole range of each decision variable to be included in the beginning of the search, but then as the number of iterations increases, the range of each decision variable decreases. Coello *et al.* [34] compared the performance of the MOPSO with the mutation operator against three other MOO algorithms, namely NSGA-II, Micro-GA [29] and pareto archived evolution strategy (PAES) [98] on five constrained MOO benchmark functions. The results of the study indicated that MOPSO with the mutation operator was the only MOO algorithm able to find solutions along the full extend of the POF for all benchmark functions.

6.4 Cooperative-coevolution Evolutionary Algorithm

CCEA, introduced by Tan *et al.* [147], is a multi-population MOO algorithm. The basic CCEA is discussed in Section 6.4.1 and the initialisation of CCEA is discussed in Section 6.4.2. Section 6.4.3 discusses the evaluation of individuals and the calculation of the niche count. Rank assignment of the individuals are discussed in Section 6.4.4.

Various genetic operators performed on the individuals are discussed in Section 6.4.5. Section 6.4.6 discusses the extending operator that is used to increase the diversity of the found non-dominated solutions.

6.4.1 CCEA

CCEA is a co-evolutionary MOO algorithm. Co-evolution can be classified into two main categories, namely competitive co-evolution and cooperative co-evolution [147]. The goal of competitive co-evolution is to obtain more competitive individuals through competitive interaction with one another (similar to the predator versus prey scenario in nature). The goal of cooperative co-evolution is to obtain better individuals through cooperation or collaboration between various independently evolving species or populations. CCEA incorporates both categories of co-evolution. The steps of the basic CCEA are listed in Algorithm 7 and discussed in more detail below.

Algorithm 7 CCEA

```
1.  create and initialise random sub-populations
2.      while stopping condition has not been reached
3.          for each parent sub-population
5.              for each individual in sub-population
6.                  evaluate the individual
7.                  update the archive
8.              for each individual in sub-population
9.                  assign rank to individual
10.                 calculate the niche count of individual
11.             perform genetic operators
12.         perform the extending operator
```

6.4.2 Initialisation

Before the algorithm starts, the sub-populations are created. The number of sub-populations are equal to the number of decision variables of the MOOP. Each sub-population is assigned one decision variable to optimise.

6.4.3 Evaluation of Individuals

Since each sub-population only optimises one decision variable, each individual in a sub-population is only a sub-component of a solution of the MOOP. Therefore, in order to evaluate the fitness of an individual, the individual is combined with representatives from other sub-populations to obtain a complete solution. The best individual in each population is selected as a representative.

The non-dominated solutions found so far by the algorithm is stored in an archive to preserve elitism. Once an individual is combined with a representative from each of the other sub-populations, the combined vector is evaluated against the solutions in the archive, and

- if the archive is empty, the combined objective vector is placed in the archive.
- if the archive is not empty:
  - if the combined objective vector is dominated by any of the solutions in the archive, the combined objective vector is not added to the archive.
  - if the combined objective vector dominates solutions in the archive, the dominated solutions are removed from the archive and the combined objective vector is added to the archive.
  - if the combined objective vector is non-dominated with regards to all solutions in the archive, and
    - the archive is not full, the combined objective vector is added to the archive.
    - the archive is full, niche count (fitness sharing) is used to determine whether a solution in the archive is replaced with the combined objective vector, and if so, which solution in the archive is replaced with the combined objective vector.

Niche count indicates how many solutions are within a specified distance of an individual. Therefore, a high niche count indicates that the individual is in a crowded area. The solution with the highest niche count will be selected to be replaced by a new solution if the archive is full.

Improper values for the radius, σ_{share}, leads to a bad distribution of solutions. However, it is not a trivial task to determine a good value for σ_{share} if the shape of the POF is unknown. Therefore, Tan *et al.* [147] calculated the niche count in a normalised objective space, where the objective space is normalised at each generation with an estimation of the ranges of the objective space. The ranges of the objective space are estimated according to the objective values in the archive at the specific generation.

6.4.4 Rank Assignment

The combined objective vector cannot be used directly as an individual's fitness in CCEA since the individual's population only optimises one decision variable. However, the combined objective vector can be used to evaluate how well an individual cooperates with other sub-populations to produce good solutions. This evaluation is done with a canonical Pareto-ranking scheme [147], where individuals are ranked according to the number of individuals in the archive that dominates the individual in objective space.

6.4.5 Genetic Operators

Tournament selection is performed in each sub-population to select parents for cross-over to produce offspring. Tournament selection is based on the individuals' rank, and if more than one individual have the best rank, the individual with the lowest niche count wins the tournament. Uniform cross-over is applied to the parents to produce offspring and bit-flip mutation is performed on the offspring.

6.4.6 Extending Operator

In order to improve the diversity of solutions, an extending operator is introduced. In CCEA, the archive member with the lowest niche count, and therefore in the least crowded region of the archive, is cloned and copied to the sub-populations. Copying this archive member to the sub-populations increases the chance that the archive member's components are selected into the mating pool and thereby attracting sub-populations to less explored regions of the search space. The steps of the extending operator are listed in Algorithm 8.

Algorithm 8 CCEA Extending Operator

```
1. if the archive is full
2.     for each solution in archive
3.         calculate the niche count
5.         find the solution with the lowest niche count, $\mathbf{p}_{nc_l}$
6.             for each sub-population
7.                 for $n$ times
8.                     randomly select an individual
9.                     replace selected individual with cloned copy of $\mathbf{p}_{nc_l}$
```

The number of cloned copies that are used is set to one. Tan *et al.* [147] compared the performance of CCEA against five MOEAs on three MOO benchmark functions. The results indicated that CCEA achieved the best overall performance with regards to converging to the true POF, as well as obtaining an even spread of solutions along the found POF.

One disadvantage of CCEA is that the number of sub-populations increases linearly as the number of decision variables increases. Therefore, Tan *et al.* [148] proposed a distributed CCEA.

6.5 Summary

This chapter provided a brief summary of important contributions to the field of MOO. Furthermore, three MOO algorithms required as background for later chapters in the thesis were discussed. NSGA-II, an improved version of NSGA, is a MOGA that uses non-domination and preserves elitism through a selection procedure. MOPSO is one of the first PSO algorithms extended for MOO and preserves elitism through an archive. CCEA is a multi-population MOO algorithm that uses cooperative co-evolution. Each population only optimises one decision variable of the MOOP and then cooperates with the other sub-populations to produce good solutions.

The next chapter discusses vector-evaluated approaches that were used to solve MOOPs.

Chapter 7

Population-based Multi-objective Vector Evaluated Approaches

"To adapt, is to move ahead." – Byron Pulsifer

This chapter discusses vector evaluated CI approaches that were introduced to solve MOOPs. Section 7.1 discusses VEGA, one of the first algorithms solving MOO without aggregating the objective functions to change the MOOP into a SOOP. VEPSO, based on the idea of VEGA but using PSO, is discussed in Section 7.2. Section 7.3 discusses another modification of VEGA, namely VEDE, where DE is used instead of GA. A hybrid algorithm based on the concept of VEPSO using different types of sub-algorithms is discussed in Section 7.4. Finally, a summary is provided in Section 7.5.

7.1 Vector Evaluated Genetic Algorithm

This section discusses the VEGA algorithm, introduced by Schaffer [131], which can be seen as the first implementation of a MOEA [31, 40]. The main steps of VEGA performed at each iteration when solving a MOOP with n_o objectives, are illustrated in Figure 7.1. These steps are:

1. the population is shuffled, the shuffled individuals are randomly divided into n_o sub-populations, and each sub-population is assigned a different objective function.

2. fitness values are assigned to individuals in each sub-population according to the sub-population's objective function. Individuals from each sub-population are selected for the mating pool using proportionate selection (refer to Section 5.2.3).
3. cross-over and mutation are applied to the mating pool to produce the next generation's population.

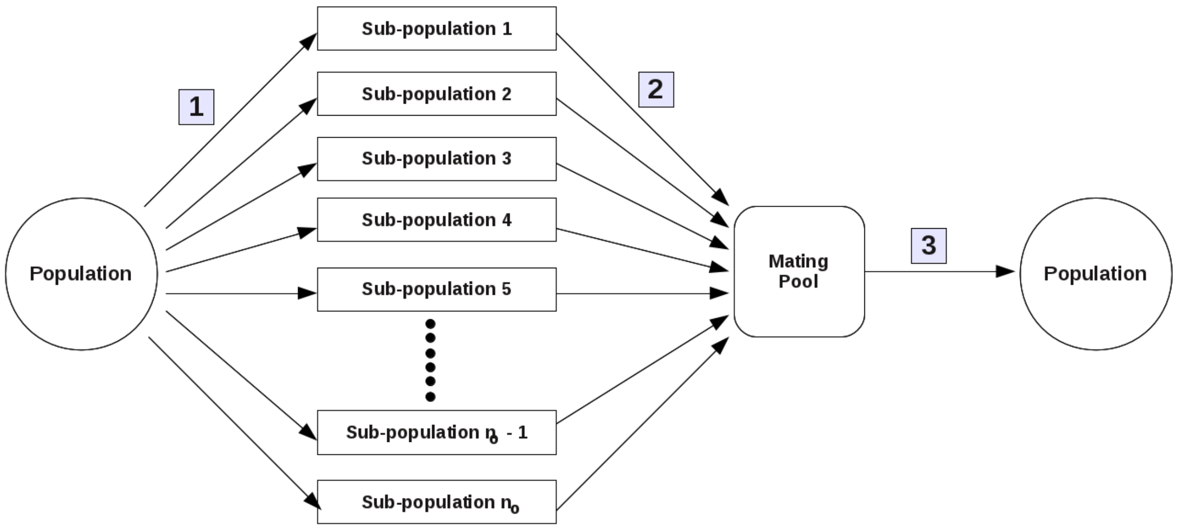


Figure 7.1: Steps of the VEGA algorithm at each generation

Each sub-population optimises one objective function and the knowledge between the different sub-populations are shared through cross-over and mutation applied to the mating pool containing individuals selected from each sub-population. However, since the selection operator is only performed per sub-population, the fitness of the individuals that are selected and placed in the mating pool are only measured based on one objective. This leads to preference of individuals that perform well with regards to the specific objective function assigned to the sub-population that the individual belongs to. Therefore, solutions that do not excel in one particular objective function, but that perform reasonably well for all objective functions (referred to as *middling* solutions), are disregarded during the selection process. This leads to an approximated POF with solutions in only certain areas of the POF. Therefore, the found POF will have a low diversity or spread of solutions. Initially, Schaffer expected that cross-over and mutation performed on a mating pool that contains solutions from all sub-populations will eliminate this problem. However, results indicated that even when using this mating pool, VEGA still did not find the middling solutions. Therefore, Schaffer [131] proposed two changes to VEGA to overcome this problem, namely:

- using a heuristic selection preference for non-dominated individuals: In stead of only measuring an individual's fitness according to one objective function, all objectives are taken into account and only non-dominated individuals can be selected for the mating pool.
- encouraging cross-breeding amongst the sub-populations by introducing a mate selection heuristic.

More information with regards to these two changes to VEGA can be found in [40].

7.2 Vector Evaluated Particle Swarm Optimisation Algorithm

This section discusses the VEPSO algorithm. The original VEPSO algorithm is discussed in Section 7.2.1. Extensions made to the original VEPSO algorithm for this thesis are discussed in Section 7.2.2.

7.2.1 Original VEPSO Algorithm

The VEPSO algorithm, inspired by VEGA [131], was introduced by Parsopoulos *et al.* [125]. Parsopoulos *et al.* [124] compared the performance of VEGA and VEPSO and found that VEPSO outperforms VEGA.

The VEPSO algorithm consists of two layers, namely a top layer that manages the sub-swarms and a lower layer that contains the sub-swarms. At the top layer the algorithm manages the sharing of knowledge between the swarms. At the lower layer, each sub-swarm optimises its assigned objective function.

Low-level Tasks

This section discusses the task of guide update approaches that are performed by the sub-swarms, i.e. on the lower level of the VEPSO algorithm.

The search process of VEPSO is driven through local and global guides. The local guides, which are actually the personal bests (or pbests), contain information about the particles' own experience with regards to a single objective. On the other hand, the global guides, which are the global bests (or gbests), contain information obtained by a pre-defined neighbourhood of particles with regards to another objective. A knowledge sharing topology determines which objective's gbest is used.

The original VEPSO articles [124, 125] do not indicate whether Pareto-dominance was used for the guide updates. Therefore, it is assumed that the original version of VEPSO updates the guides according to the particles' fitness with regards to only one objective, i.e. the objective that the specific swarm is optimising. To solve DMOOPs, Pareto-dominance was added to the VEPSO algorithm, as discussed in Section 9.3.

Top-level Tasks

This section discusses knowledge exhange between the sub-populations that are performed at the top level of the VEPSO algorithm.

The number of sub-swarms is equal to the number of objectives of the optimisation problem and each swarm optimises only one objective function. Knowledge of best solutions is then shared with the other swarms. This shared knowledge, contained in the global guide of another swarm, is then used to update the velocity of the particles:

$$\begin{aligned} S_k.v_{ij}(t+1) &= wS_k.v_{ij}(t) + c_1 r_{1j}(t)(S_k.y_{ij}(t) - S_k.x_{ij}(t)) \\ &+ c_2 r_{2j}(t)(S_s.\hat{y}_{ij}(t) - S_k.x_{ij}(t)) \end{aligned} \quad (7.1)$$

where $k = 1, \ldots, m$ represents the index of the respective swarm, $v_{ij}(t)$ and $x_{ij}(t)$ represent the j-th dimension of the velocity and position of particle i at time t respectively, w is the inertia weight, $S_s.\hat{\mathbf{y}}_i$ is the global best of the s-th swarm, c_1 and c_2 are respecively the cognitive and social coefficients, $\mathbf{r}_1, \mathbf{r}_2 \in [0,1]^n$, and n is the dimension of the search space.

The index, s, of the swarm from which knowledge is obtained, is selected based on a knowledge sharing topology. The original VEPSO used a ring topology where s is selected according to Equation (7.2), where

$$s = \begin{cases} M & \text{for } j = 1 \\ j-1 & \text{for } j = 2, \ldots, M \end{cases}$$

Therefore, VEPSO has two topologies, namely:

- the topology of the swarms that is used to exchange knowledge between the different swarms.
- the topology of the particles in each swarm that is used for the global guide update.

Since each swarm optimises only one objective function, an increase in the number of objective functions of a MOOP will result in a linear increase in the number of required swarms, resulting in an increase in computational complexity. To overcome this problem, Parsopoulos and Vrahatis presented a VEPSO design that enables VEPSO to be executed in parallel [124]. Each swarm's velocity update is performed in isolation from the other swarms. Before each parallel execution step, the state of knowledge being shared is recorded and the recorded knowledge is then used to update the particles' velocities.

7.2.2 Extensions to the VEPSO Algorithm

This section discusses extensions to the VEPSO algorithm. Extensions introduced at the sub-swarm level as well as the top level are presented. At the sub-swarm level the management of boundary constraint violations are introduced. At the top level of the algorithm knowledge sharing approaches between the sub-swarms and the management of an archive are introduced.

Low-level Tasks

The original VEPSO articles [124, 125] do not indicate whether boundary constraint violations are managed. Therefore, it is assumed that the original version of VEPSO does not manage the violation of boundary constraints. This section discusses an additional task performed by the sub-swarms, namely managing particles that move outside the search space during the optimisation process.

Some of the approaches that have been used to pull a particle back into the search space and proposed to be used for VEPSO [77] are:

- *Clamping*, where each particle that violates a specific boundary of the search space is placed on or close to the violated boundary of the search space [122]. Clamping is defined as:

$$\begin{aligned} \text{if } \mathbf{x}(t+1) &> \mathbf{x}_{max}, \text{ then } \mathbf{x}(t+1) = \mathbf{x}_{max} - \epsilon \\ \text{if } \mathbf{x}(t+1) &< \mathbf{x}_{min}, \text{ then } \mathbf{x}(t+1) = \mathbf{x}_{min} \end{aligned} \tag{7.2}$$

  with ϵ a very small positive number.
- *Deflection*, where the velocity's direction of the violated dimension is inverted, thereby causing a bouncing effect off the bounds. The deflection approach is defined as:

$$\begin{aligned} \text{if } x_i(t+1) > x^i_{max}, \text{ then } \quad x_i(t+1) &= x^i_{max} - (x_i(t+1) - x^i_{max})\%(x^i_{max} - x^i_{min}) \\ \text{and} \quad & v_i(t+1) = -v_i(t) \\ \text{if } x_i(t+1) < x^i_{min}, \text{ then } \quad x_i(t+1) &= x^i_{min} + (x^i_{min} - x_i(t+1))\%(x^i_{max} - x^i_{min}) \\ \text{and} \quad & v_i(t+1) = -v_i(t) \end{aligned} \tag{7.3}$$

  where x_i, x^i_{min} and x^i_{max} are the i-th dimension of $\mathbf{x}$, $\mathbf{x}_{max}$ and $\mathbf{x}_{min}$ respectively.
- *Per element re-initialisation*, where each dimension of the particle's position that violates the boundary constraint is re-initialised to a random valid value [122]. Therefore, the dimensions of the position that is valid remains the same. Per element re-initialisation is defined as:

$$\begin{aligned} \text{if } x_i(t+1) > x^i_{max}, \quad \text{then } \quad x_i(t+1) = rand(x^i_{min}, x^i_{max}) \\ \text{if } x_i(t+1) < x^i_{min}, \quad \text{then } \quad x_i(t+1) = rand(x^i_{min}, x^i_{max}) \end{aligned} \tag{7.4}$$

- *Periodic*, which is similar to the deflection approach, but the violating particle is placed near the lower boundary of the dimension if it violates the upper boundary of the dimension and vice versa [162]. The periodic approach is defined as:

$$\begin{aligned} \text{if } x_i(t+1) > x^i_{max}, \text{ then } x_i(t+1) &= x^i_{min} - (x_i(t+1) - x^i_{max}) \\ &\quad \%(x^i_{max} - x^i_{min}) \\ \text{if } x_i(t+1) < x^i_{min}, \text{ then } x_i(t+1) &= x^i_{max} - (x^i_{min} - x_i(t+1)) \\ &\quad \%(x^i_{max} - x^i_{min}) \end{aligned} \tag{7.5}$$

- *Random*, which re-initialises a particle's position to a valid position within the search space if it violates the boundaries of the search space [79, 162]. Therefore, in contrast to the per element re-initialisation approach, all dimensions are re-initialised and not only the violating dimensions. Random is defined as:

$$\begin{aligned} &\text{if } \mathbf{x}(t+1) > \mathbf{x}_{max}, \quad \text{then } \mathbf{x}(t+1) = rand(\mathbf{x}_{min}, \mathbf{x}_{max}) \\ &\text{if } \mathbf{x}(t+1) < \mathbf{x}_{min}, \quad \text{then } \mathbf{x}(t+1) = rand(\mathbf{x}_{min}, \mathbf{x}_{max}) \end{aligned} \tag{7.6}$$

- *Re-initialisation*, where a particle that violates the bounds of the search space has its position re-initialised to a valid position within the search space, its velocity set to zero, and its local guide set to the particle's new position [122].
- *Unconstrained*, where no clamping is performed and particles are free to move outside the search space. However, only valid positions are selected as the local guide of a particle.

Top-level Tasks

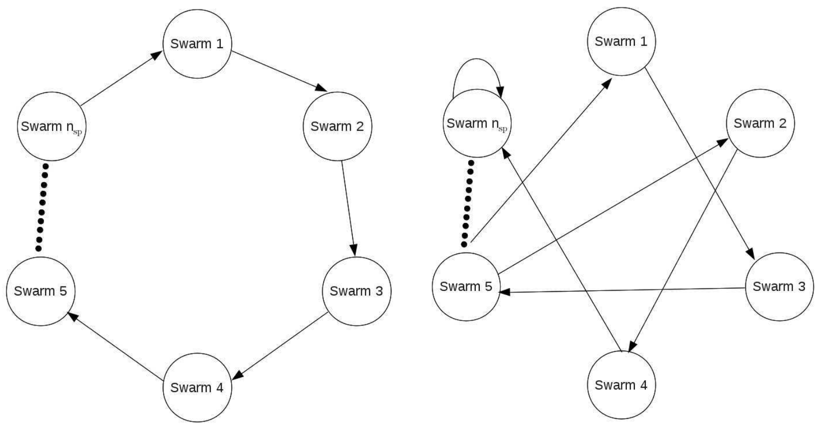
This section discusses additional knowledge exhange approaches and the task of archive management that are performed at the top level of the DVEPSO algorithm.

Knowledge Sharing

The original VEPSO algorithm used the ring topology. Recently, an alternative topology has been proposed, namely a random topology [71][1]. If a random topology is used, s is selected randomly from $[1, m]$. These two VEPSO topologies are illustrated in Figure 7.2, with a ring topology on the left and a random topology on the right. With a random topology, s can be the index of another swarm (for example swarms 1-5 in Figure 7.2

[1]Greeff is the maiden name of M. Helbig.

(b)) or from the swarm itself (for example swarm n_o in Figure 7.2 (b)), i.e. a swarm can use its own *gbest* value to update its particles' velocity. The selection of index s can be done at the beginning of an algorithm run, or after a specified number of iterations. Using a different s every now and again may guide the particles to different parts of the POF since information about another objective is provided. However, if the index of s is changed too often it may slow down the convergence of the algorithm by guiding the particles in a new search direction before optimal solutions were found.



(a) Ring Topology (b) Random Topology

Figure 7.2: Topologies of VEPSO algorithm

Inter- and Intra Swarm Speciation

According to Matthysen and Engelbrecht [116], the pbest update approach used by VEPSO, together with a ring or random topology for the gbest update, may lead to inter- and intra-swarm speciation defined below.

If the gbest is not updated often, the particles' trajectories will end near the gbest. The gbest will cause the particles to converge to a point that is a weighted average between the particle's pbest and the swarm's gbest [27, 63]. This will prevent particles from exploring the entire POF and therefore only sub-regions of the POF will be found. This problem, similar to the problem experienced by VEGA, is referred to as inter-swarm speciation.

One approach to overcome inter-swarm speciation, is to update the gbest more frequently using the random swarm topology (refer to Figure 7.2 (b)). However, the random swarm topology can lead to intra-swarm speciation. Intra-swarm speciation occurs when the randomly selected swarm index results in a swarm using its own gbest frequently to update its particles' velocity. In this situation, the pbest of each particle in the swarm is updated according to only the objective function assigned to the swarm. Then, if the swarm's own gbest is used, the swarm obtains more information about its own objective function being optimised than other swarms that do not use their own gbest. This will lead to exploitation of the swarm's knowledge about its objective function, that may lead to more updates of the swarm's gbest. Even though there is the danger of intra-swarm speciation, the random swarm topology overcomes the problem of inter-swarm speciation and therefore may lead to a more diverse set of solutions being found.

Another approach that can be followed to overcome inter-swarm and intra-swarm speciation is to use information about the other objective functions when updating the pbest and gbest. In this case, each swarm still only optimises one objective function and shares knowledge with each other as discussed above. However, when updating the pbest and gbest, Pareto-dominance is used, taking into account the whole MOOP, and only non-dominated solutions are allowed to become a pbest or gbest.

Archive Management

As the VEPSO algorithm optimises the MOOP, the non-dominated solutions found so far are stored in an archive. Due to limited resources, the size of the archive is normally limited. The following approach is normally used by MOO algorithms to manage the archive:

- if a new non-dominated solution is dominated by any solution in the archive, the new solution is not added to the archive.
- if the new solution dominates any solutions in the archive, the dominated solutions are removed from the archive and the new solution is added to the archive.
- if the new solution is non-dominated with regards to all other solutions in the archive and there is space in the archive, add the new solution to the archive. However, if the archive has reached its maximum size, an approach has to be followed to determine which solutions to remove from the archive. One approach is to remove solutions from the more dense areas of the found POF [6]. This

approach ensures that a diverse set of solutions are kept.

7.3 Vector Evaluated Differential Evolution Algorithm

Inspired by VEGA [131], Parsopoulos *et al.* [123] introduced the vector evaluated differential evolution (VEDE) algorithm. VEDE is a multi-population DE algorithm. DE, introduced by Storn and Price [142], is a direct search method that utilises n_s, n-dimensional vectors. VEDE, similar to VEGA and VEPSO, assigns a single objective function to each sub-population. Therefore, the fitness of the individuals of each sub-population is evaluated against only the objective function that is assigned to that sub-population.

However, contrary to VEGA and VEPSO, VEDE knowledge sharing takes place through not only the usage of the knowledge, but through the immigration of the best individuals from one population to another.

Similar to VEPSO, an increase in the number of objective functions will require a linear increase in the number of populations of the VEDE algorithm. Therefore, Parsopoulos *et al.* [123] presented a parallel implementation of VEDE. After each generation, the best individual of each sub-population is sent to the master node. The master node then sends the migrating individual to the appropriate sub-population, i.e. the next sub-population according to the defined ring topology.

The performance of VEDE was compared against VEGA. Parsopoulos *et al.* [123] found that VEDE outperforms VEGA. However, the study also revealed that VEDE is sensitive to population size, especially when the number of individuals in a sub-population becomes small (less than 20) [123].

7.4 Hybrid Vector Evaluated Algorithm

According to Grobler and Engelbrecht [73], VEDE exploits good solutions at the cost of diversity, while VEPSO explores the search space more than VEDE, resulting in more diverse solutions. Therefore, in order to exploit the good qualities of both VEDE and VEPSO, Grober and Engelbrecht introduced the vector evaluated differential evolution particle swarm optimisation (VEDEPSO) algorithm as a hybridisation of VEPSO and VEDE.

Similar to VEPSO and VEDE, each sub-population of VEDEPSO is assigned one objective function to optimise. VEDEPSO uses the random topology (refer to Figure 7.2 (b)) for knowledge sharing. For bi-objective problems, one sub-population is a VEPSO and the other sub-population is a VEDE [73]. However, the authors do not specify how to handle more than two sub-populations.

Grobler and Engelbrecht [73] compared VEPSO, VEDE and VEDEPSO on five MOO benchmark functions and found that VEDEPSO outperformed VEPSO and VEDE on four out of the five MOOPs. Furthermore, the results of the study indicated that VEDEPSO performed significantly better on all benchmark functions than the worst performance obtained by either VEPSO or VEDE.

7.5 Summary

This chapter discussed vector evaluated approaches for solving MOOPs. VEGA [131], introduced by Schaffer, can be seen as the first EA used to solve MOO without aggregating the objective functions. Each sub-population of VEGA optimises only one objective function. The knowledge of the various sub-populations are shared through mutation and cross-over on a mating pool consisting of individuals selected from each sub-population. One problem with VEGA is the occurrence of inter-swarm speciation, namely that it favours solutions that excel in one objective. To overcome the problem of speciation, variations to the orignal VEGA algorithm were proposed.

Inspired by VEGA, Parsopoulos *et al.* [125] introduced VEPSO. Similar to VEGA, VEPSO divides the swarm into sub-swarms and each sub-swarm only optimises one objective function. The various sub-swarms share their knowledge through the velocity update of the particles. If the standard PSO pbest update is used together with a ring topology, VEPSO may struggle with inter-swarm speciation, similar to VEGA. However, the problem of inter-swarm speciation can be overcome by using a different knowledge exchange strategy.

Another algorithm was also inspired by VEGA and introduced by Parsopoulos *et* *al.* [123], namely VEDE. Similar to VEGA and VEPSO, each sub-population of VEDE optimises only one objective function. However, contrary to VEGA and VEPSO, knowledge sharing takes place through not only the usage of the knowledge, but through the immigration of the best individuals from one sub-population to another.

Since both VEDE and VEPSO have good qualities, Grobler and Engelbrecht [73] introduced an algorithm that hybridises both of these vector evaluated approaches, called VEDEPSO. Each sub-population of VEDEPSO is either a VEDE or VEPSO algorithm. The study by Grobler and Engelbrecht indicated that the hybridised algorithm outperformed the other two vector-evaluated approaches on five MOOPs.

The next chapter discusses population-based algorithms that were introduced to solve dynamic MOOPs.

Chapter 8

Population-based Dynamic Multi-objective Optimisation Algorithms

"The art of life is a constant readjustment to our surroundings."
–K. Okakaura

DMOOPs are optimisation problems with multiple objectives with at least one objective changing over time. The objectives are in conflict with one another and therefore the problem does not have a single solution, as is the case with DSOOP. A DMOOP, similar to a MOOP, has a set of trade-off solutions called the POF. Therefore, in order to solve a DMOOP, an algorithm must be able to track the changing POF over time.

This chapter discusses population-based algorithms proposed to solve DMOOPs. Section 8.1 provides an overview of DMOO algorithms that have been proposed in the literature and a summary is provided in Section 8.2.

8.1 Dynamic Multi-objective Algorithms

This section discusses algorithms that have been proposed for DMOO. Algorithms that aggregate the objective functions of the DMOOP to create a DSOOP are not considered.

All acronyms used in this section for benchmark functions and performance measures were defined in Chapters 3 and 4 respectively.

Section 8.1.1 discusses SMOO algorithms that were adapted to solve DMOOPs. New CI algorithms that were introduced for DMOO are discussed in Section 8.1.2. Section 8.1.3 discusses approaches that are used to convert a DMOOP into multiple static MOOPs (SMOOPs). Generic extensions that can be applied to any DMOO algorithm are discussed in Section 8.1.4. Section 8.1.5 discusses prediction-based approaches where knowledge of previous environments is used to predict the new POS or POF.

8.1.1 Multi-objective Optimisation Algorithms adapted for Dynamic Multi-objective Optimisation

One of the first algorithms proposed to solve DMOOPs without using the weighted sum approach to aggregate the objective functions into a DSOOP, was a hybridised minimal cost evolutionary deterministic algorithm (HMCEDA) introduced by Farina *et al.* [58]. With the hybrid algorithm an (1+1) evolution strategy (ES) is used for global optimization of the DMOOP [57]. An (1+1) ES is an EA where each iteration applies Gaussian mutation to one parent to create one offspring, i.e. a random value from a Gaussian distribution is added to each element of a parent's vector to create an offspring [9]. Once the (1+1) ES starts to converge (determined by comparing the decision variable values from two consecutive iterations), a gradient-based algorithm or a simplex Nelder Mead search algorithm [120] is used. HMCEDA was evaluated on the FDA DMOOPs. For FDA1 and FDA2, the algorithm tracked the changing POF well over time and converged quickly to the new POF after a change in the environment occurred. However, for FDA3, HMCEDA struggled to converge towards the changing POF and struggled to find a diverse set of solutions. For FDA4, the algorithm converged reasonably well to the new POF after each change in the environment. However, for FDA5 where the density of the solutions change over time, HMCEDA struggled to maintain a diverse set of solutions [58]. According to Farina *et al.* [58] the results from the study indicate that HMCEDA should use an EA with better performance, such as NSGA-II, for the global optimization.

Many algorithms used to solve SMOOPs were adapted for DMOO. Avdagić *et al.*

extended MOGA to solve DMOOPs [2]. MOGA was the first multi-objective GA introduced by Fonseca and Flemming [60] and uses Pareto-ranking. The advantages of MOGA is a simple fitness assignment and the easy application of MOGA to other types of optimisation problems (such as combinatorial optimisation problems), since niching takes place in objective space [40]. Furthermore, MOGA maintains a diverse set of solutions. One problem with averaging the fitness values for Pareto ranking is that all solutions in a specific front have the same fitness. For the front with Pareto rank 1 it does not matter that all solutions in the front have the same assigned fitness, since all the solutions are non-dominated. However, for the other fronts, when all solutions have the same assigned fitness, the search may be biased towards unwanted solutions in the search region. If a solution of a lower front (thereby being dominated by less solutions) is located in a crowded region, its niche count will be large and therefore the solution will obtain a lower average fitness than a solution of a higher front (being dominated by more solutions) in a less crowded region. This can cause the search to be biased towards the solution in the higher front, instead of the solution in the lower front [40]. MOGA was evaluated on modified DTLZ functions and its performance was measured using C_o and the HV. MOGA converged well towards POF and found a diverse set of solutions [2].

NSGA-II, one of the most successful MOO algorithms that are a benchmark for MOO research, was adapted for DMOO by Deb *et al.* [46]. A few solutions are randomly selected and re-evaluated and if there is any change in the objective values, it is assumed that a change in the environment has occured. If a change has been detected, the population is re-evaluated. To increase diversity, the following two approaches are used after a change in the environment took place:

- A percentage of the population is replaced with randomly created individuals. This approach is referred to as dynamic NSGA-II (DNSGA-II)-A. Since this approach introduces new solutions, it increases the search space that is covered by the population and may lead to improved performance in environments with severe changes [46]. However, if the environment has small changes, the new individuals may misguide the search to new areas of the search space that is far away from the current optima.
- A percentage of the population is replaced with mutated solutions of randomly

selected existing solutions. This approach is referred to as DNSGA-II-B. Since the new individuals are related to the individuals of the current population, this approach may be beneficial in environments with small changes [46]. However, in environments that change severely, the new individuals may guide the search in new areas that are far away from the new optima. This may cause the algorithm to become stuck in old optima.

Deb *et al.* [46] investigated the performance of the DNSGA-II [47, 42] algorithms on a modified version of FDA2 and measured the algorithms' performance using the HVR. The results indicated that the performance of DNSGA-II deteriorated when the frequency of change increased. Furthermore, when the number of random solutions introduced after an environmental change were increased, the performance of DNSGA-II-A deteriorated. When the number of mutated individuals added to the population after a change occurred was increased, the performance of DNSGA-II-B decreased slightly. However, DNSGA-II-B performed better than DNSGA-II-A. This may be explained by the fact that even though the shape of the POF changed from convex to concave over time, the new POF was not far from the previous POF in objective space. Furthermore, adding a small percentage of either random or mutated individuals after a change in the environment occurred, lead to better performance than not adding any new solutions at all. The study also indicated that almost any percentage of the population can be mutated and NSGA-II-B still performed well. However, with DNSGA-II-A, 20-40% of random individuals after a change lead to good performance [46].

In order to determine the efficiency of various MOO algorithms solving DMOOPs, Mehnen *et al.* [117] compared the performance of three MOEAs, namely NSGA-II [47, 42], SPEA2 (SPEA2) [170] and multiple single objective Pareto sampling (MSOPS) [86]. MSOPS is a stochastic population-based algorithm that does not incorporate Pareto-dominance. MSOPS uses a weighted min-max aggregation of the objectives and a ranking scheme according to weight vectors referred to as targets. However, contrary to conventional linear aggregation approaches, weighted min-max is capable of finding solutions on non-convex parts of the POF. The performance of these three MOEAs were compared solving eight DMOOPs, namely DSW1, DSW2, DTF and the FDA DMOOPs. The results indicated that, when simulated binary crossover (SBX) [43] was used together with polynomial mutation (PM) [39], all MOEAs were capable of tracking the changing POS when solving the Type I DMOOPs. Furthermore, MSOPS concentrated more on the central region of the POF when solving DMOOPs with a small severity of change. When DE variation operators were used instead of SBX and PM, the population's diversity collapsed and then the operators were no longer capable of creating new solutions. Therefore, the MOEAs got stuck. None of the MOEAs were able to track the changing POF when the environment changed at every iteration. However, MOEAs using SBX and PM converged towards the POF when the environment changed every 25 generations or less frequent than every 25 iterations. NSGA-II converged faster than MSOPS, which in turn converged faster than SPEA2. For FDA4 and FDA5, MSOPS converged quicker towards the POF. NSGA-II and SPEA2 required more function evaluations in order to converge towards the POF. Both NSGA-II and SPEA2 experienced a decrease in selection pressure, since using the Pareto-dominance relation resulted in a number of incomparable solutions. Once again, when optimising FDA4 and FDA5, MSOPS concentrated on certain parts of the POF, where as NSGA-II and SPEA2 obtained a diverse set of solutions. Similar trends were observed with FDA3 and FDA5 where the density of the solutions changed over time. Therefore, Mehnen *et al.* [117] concluded that if a population contains many incomparable individuals, the Pareto dominance based algorithms cannot guarantee further convergence. For FDA2 the POF changes in shape over time, from a convex POF to a non-convex POF. For DTF, the POF changes structure over time, since the number of continuous sections of the disconnected POF vary over time. MSOPS struggled to adapt to a change in either the shape or structure of the POF over time. NSGA-II obtained the best performance for FDA2. On the other hand, SPEA performed well when solving DTF.

An artificial immune system (AIS) algorithm, the clonal-selection algorithm for DMOO (CSADMO), was introduced by Shang *et al.* [135]. CSADMO uses clonal selection [14], a self-adaptive (dynamic) process of the immune system. Non-uniform mutation [118] is used, with the search space being searched uniformly with large jumps during the first few iterations, but during the later generations the search is more local with small jumps. Furthermore, crowding distance [42] is used to increase the diversity of the solutions. The performance of CSADMO was compared against HMCEDA of Farina *et al.* [58] using

FDA3 and FDA5. However, the authors do not mention which frequency (τ_t) and severity of change (n_t) were used for the DMOOPs. The results indicate that both algorithms performed well with regards to convergence and diversity. However, CSADMO outperformed HMCEDA with regards to both convergence and diversity on both DMOOPs. Convergence was measured using GD in the objective space, but diversity was only determined using a graphical representation of the found POFs. Furthermore, the authors used the results of HMCEDA directly from [58]. This may lead to misleading results, since the same set of sampling points for the true POF is not used for the calculation of the GD values for both algorithms.

Zhang [163] also proposed an AIS, MOIA, to solve DMOOPs. Immune operators were proposed to enable the algorithm to adapt to dynamic environments. Furthermore, Zhang proposed an environment recognition rule that is based on previous environmental information. The environment recognition rule classifies the current environment as either a new, similar or an identical environment in comparison to the previous environment. This classification is used to determine which operators to use. MOIA was compared against NSGA-II and SPEA2 using FDA2, FDA4 and a real-world problem. The algorithms' performance was measured using the C-metric and Schott's Spacing measure. The results indicate that NSGA-II outperformed SPEA2 for FDA2 and FDA4, but was outperformed by SPEA2 when solving the real-world problem. MOIA outperformed NSGA-II and SPEA2 on all three problems.

An orthogonal multi-objective EA for DMOO (DOMOEA) [160] was introduced by Zeng *et al.* DOMEA uses two types of cross-over operations, namely orthogonal cross-over that is based on orthogonal design [160] and linear cross-over [158]. One of these cross-over operators are randomly selected and applied to two randomly selected parents to produce offspring. If the number of non-dominated solutions are more than the population size, a clustering technique is used to select individuals for the new population. DOMOEA was evaluated on FDA1, FDA2 and FDA3, with $n_t = 10$ and $\tau_t = 150$. The algorithm's performance was measured using GD and Spread as defined by Deb [42]. The results indicate that DOMOEA successfully converged to the POF and found a diverse set of solutions.

Zhen [165] also proposed a MOEA adapted for DMOO (DMOEA). When a change is detected in the environment, hypermutation [28] is used to preserve a certain number of elitist individuals, while the rest of the individuals are replaced by randomly created new individuals. Furthermore, DMOEA uses geometrical Pareto-selection to manage the archive. This approach selects an auxiliary point that is far away from POF^*. Each solution in POF^* is connected to the auxiliary point, enabling the calculation of a slope. If two solutions have the same slope, the solution furthest away from the auxiliary point has the best fitness (assuming minimisation). If a new solution is considered to be stored in the archive, the new solution is only compared against the solutions that are located in the same slope region as the new solution. If the new solution has a larger distance to the auxiliary point than one of the solutions in the archive that it is compared against, the new solution replaces the other solution. DMOEA was evaluated on FDA1, modified FDA2, modified FDA3, FDA4 and FDA5 using the HV. For FDA1 to FDA3, $\tau_t = 2000$ and for FDA4 and FDA5, $\tau_t = 5000$. Therefore, the environment changed only every 2000 or 5000 generations. The results indicate that DMOEA successfully tracked the changing POF and found a diverse set of solutions.

A multi-strategy ensemble MOEA, referred to as MS-MOEA, was introduced by Wang and Li [156] to solve DMOOPs. If a change is detected, each individual is either re-initialised to a random new position or re-initialised by selecting a random parent and adding values of a Gaussian distribution to all variables of the parent to create the new individual. During the search, two possible combinations of operators are used to create offspring, namely: (i) SBX and polynomial mutation, or (ii) DE operators. MS-MOEA was compared against an improved version of NSGA-II (INSGA-II) [161], FH-MOEA [154] and MS-MOEADE [156] using the FDA1, FDA2, FDA3, DMZDT functions and WYL. The performance of the algorithms were measured using IGD and the HV. The results indicate that MS-MOEA outperformed the other algorithms on FDA1 and FDA2, and obtained the second highest performance on FDA3. Furthermore, MS-MOEA outperformed the other algorithms on all DMZDT functions and WYL. Only on DMZDT4 with the slowest changing environment setting did MS-MOEA obtain the second highest rank, with INSGA-II obtaining the best performance.

Chen *et al.* [23] introduced the individual diversity multi-objective optimization evolutionary algorithm (IDMOEA) that uses diversity as an additional objective when solving DMOOPs. The first step of IDMOEA entails checking whether a change in the environment has occurred. If an environment change is detected, a new population is created and the best individuals of the population and the archive are selected for the new population. During the next step of the algorithm crossover is performed on parents (selected through binary tournament selection) to produce offspring, mutation is performed on the offspring and the new population is selected. Finally, the archive is updated by adding non-dominated individuals of the population to the archive. If there are more non-dominated individuals than the size of the archive, the individuals with a better diversity (calculated according to the diversity measure that is used as an additional objective) are added to the archive. IDMOEA was evaluated on FDA1 and FDA5. The algorithm's performance was measured using GD and entropy. The results indicate that the algorithm showed good convergence and maintained a diverse set of solutions.

Recently dynamic multi-objective gradient search (MO-EGS) [65] was adapted for DMOO by Koo *et al.* [100]. The new algorithm is called dynamic MO-EGS (dMO-EGS). A change is detected by re-evaluating solutions in the archive. When a change in the environment occurs, information with regards to the outdated archive (i.e. the archive containing solutions for the previous environment) is represented by the archive centroid and the variance of the archive centroid. The archive centroid is calculated by summing the decision vectors of each solution in the archive and then dividing by the number of archive solutions. A memory item that consists of the centroid and centroid variance is added to the external memory. If the external memory is full, the oldest memory item is replaced.

After a change in the environment has been detected, a truncation operator is used to remove solutions from the outdated archive in such a way that the most diverse portion of solutions are retained. Mutation is then applied to the retained solutions as a form of hypermutation to increase the diversity of the solutions. After mutation has been applied, the retained solutions of the archive are re-evaluated. In addition to mutation, in order to increase the exploration ability of dMO-EGS after a change has occurred, the mutation step size is reset.

dMO-EGS uses a weighted sum average of the objectives to calculate an individual's fitness during the optimisation process. The weights are created randomly and normalised in such a way that the sum of the weights are equal to one. However, Pareto dominance is used to determine which solutions are stored in the archive.

The performance of dMO-EGS was compared against two other DMOO algorithms, namely dCCEA [147] and dPAES [98] (a dynamic version of PAES). The same change response strategies are implemented in dCCEA and dPAES, namely storing and retrieving of memory items, hypermutation and re-evaluation of archive solutions. The three algorithms were tested against four benchmark functions, namely FDA1, FDA3, DIMP1 and DIMP2. For FDA1 and FDA3, both dynamic CCEA (dCCEA) and dMO-EGS performed signficantly better than dPAES with regards to both accuracy of the found POS and the spread of solutions. All three MOEAs had difficulty finding a diverse set of solutions when solving DIMP1. However, dMO-EGS obtained better results for DIMP1 than both dCCEA and dPAES. dMO-EGS obtained the best performance for DIMP2 when the landscape changes were not so severe and the frequency of change was low. When the environment changed severly, dCCEA outperformed dMO-EGS.

8.1.2 New Computational Intelligence Algorithms

A few algorithms introduced for DMOO were not merely adapted SMOO algorithms, but new types of CI algorithms. Amato and Farina [1] proposed an ALife-inspired algorithm to solve DMOOPs. The individuals of the algorithm are coded strings, similar to a GA, but the operators are based on individuals interacting in a population, i.e. the individuals can reproduce, meet each other or fight with each other. Unlike other EAs where selection takes place for the cross-over operator, no selection takes place. Each individual has the same probability to meet another individual. If a meeting does take place, another individual is randomly selected for the meeting. During the meeting the individuals either reproduce or fight. If reproduction occurs, two offspring are produced and added to the population. However, if the two individuals fight, the loser of the fight (the individual that Pareto dominates the other, or if both individuals are non-dominated, the individual with the most neighbours in a specified neighbourhood) is removed from the population. Due to the operators that are performed on the individuals, the population size varies. Since no selection is required, fitness evaluations are only performed when a meeting occurs between individuals, and therefore not necessarily for each individual at each iteration. For very complex problems, this may lead to a lower computational cost. Furthermore, the algorithm does not have to check for a change in the environment, but automatically tracks the changing POF. However, a major problem with the algorithm is slow convergence and therefore the algorithm will struggle to track the changing POF in a fast changing environment.

Goh and Tan [147] introduced a modified version of CCEA (refer to Section 6.4) to solve DMOOPs. The cooperative EA, CCEA, is extended to incorporate a competitive mechanism and this extended algorithm is referred to as the competitive-cooperative evolutionary algorithm (COEA) [67]. COEA sub-populations optimise only one decision variable. The competitive mechanism is used to determine the most suitable sub-population for each decision variable. The selected sub-population is then used during the cooperative phase of the algorithm to optimise that specific decision variable. The competitive mechanism is implemented as follows: for each decision variable, a representative solution is selected from the associated sub-population and placed in a competition pool along with the competitors that are selected from the other sub-populations. One competitor is selected randomly from each competing sub-population. Once the competition pool has been created, the competitive process is performed on the competition pool. Through the competitive process, the winning sub-population is determined for the specific decision variable and then assigned to optimise the specific decision variable.

Goh and Tan [67] extended the COEA algorithm to detect and respond to changes in the environment. This modified DMOO version of CCEA is called dynamic COEA (dCOEA). A change in the environment is detected by re-evaluating a fixed number of solutions in the archive and checking whether there is a difference between a solution's previous value and the re-evaluated value. If there is a difference, a change is detected and the competitive mechanism is started. Using the competitive mechanism after a change occurs enables the algorithm to apply existing information of the various sub-populations to the new environment. Furthermore, diversity is introduced into the sub-populations through the competitive mechanism, instead of using other diversity mechanisms, such as hypermutation. dCOEA increases diversity through the competitive mechanism by introducing a set of stochastic solutions together with the competitors of other sub-populations into the competition pool. If a stochastic solution emerges as the winning solution after the competitive process, the associated sub-population is re-initialised in the region that the winning solution has been sampled from. This approach ensures that diversity is only introduced into a sub-population if the added diversity produces better solutions than the sub-population's current individuals. The ratio between the stochastic solutions and the competitors are determined by a pre-defined parameter, SC_{ratio}. dCOEA stores the non-dominated solutions that have been found so far in an external population, also referred to as temporal memory, in addition to an archive. When a change occurs, a fixed number of solutions, R_{size}, of the archive are stored in the temporal memory, before all solutions are removed from the archive. If the temporal memory is full, the oldest set of R_{size} is removed to make place for newer solutions. To ensure that the R_{size} solutions of the archive that are now in the temporal memory do not misguide the optimisation process, these solutions are not re-inserted into the sub-populations during the generation immediately after an environmental change. The value of R_{size} determines how many solutions of each environment are stored in the temporal memory. A small R_{size} value causes that a smaller portion of solutions obtained from more environments are stored in the temporal memory. In this case dCOEA has access to information that ranges across various landscapes.

The performance of dCOEA was compared against adapted versions of a basic MOEA and CCEA [147]. Random restart and temporal memory were incorporated into MOEA and CCEA. dCOEA and the adapted MOEA and CCEA algorithms (referred to as dynamic MOEA (dMOEA) and dCCEA respectively) were evaluated on FDA1, dMOP1, dMOP2 and dMOP3. The study indicates that dCOEA outperformed dMOEA and dCCEA with regards to convergence to the true POF and found a diverse set of solutions for FDA1 and dMOP1. Furthermore, as can be expected, all three algorithms obtained better convergence and diversity for less frequent landscape changes, since the algorithms then had a longer time to converge to the POF before a change in the environment occurred. For high change frequencies where the environment changed every five or ten iterations, dCCEA outperformed dCOEA with regards to convergence to the true POF. However, dCOEA outperformed dMOEA and dCCEA with regards to convergence and diversity when the environment changed less frequently or when the environment changes were less severe. Further investigation revealed that a lower SC_{ratio}

value improved dCOEA's performance in environments that changed frequently. With regards to dMOP3, where the spread of solutions of the true POF changes over time, dCOEA outperformed dMOEA and dCCEA with regards to both convergence and diversity. However, all three algorithms struggled to obtain a diverse set of solutions when the environment changed frequently, i.e. every five or ten iterations.

Recently, Huan *et al.* [84] introduced a DMOO algorithm based on P systems or membrane computing [64], referred to as DMOAP. A membrane structure consists of a main membrane that contains other membranes. A P system is a membrane structure where the membranes contain objects that can evolve. DMOAP consists of a skin membrane and a m_{mid} membrane. The m_{mid} membrane contains $m + 1$ membranes or sub-systems. m of these sub-systems optimise only one objective function and the other sub-system optimises all objective functions simultaneously. Each sub-system has its own sub-population and works similar to a single-objective EA. The m_{mid} membrane collects the chromosomes that are produced by the sub-systems and sends the non-dominated solutions to the skin membrane. The skin membrane then selects the best trade-off solution(s). DMOAP is tested on a control problem of a time-varying unstable plant. The results indicate that DMOAP effectively solved the control problem.

Many MOEAs were adapted for DMOO. However, only a few PSO-based algorithms were proposed to solve DMOOPs. Lechuga [102] proposed the first PSO-based DMOO algorithm by extending MOPSO [33] (refer to Section 6.3) for DMOO. To detect changes in the environment, sentry particles [22] are used. A specified number of particles, called sentry particles, are randomly selected and re-evaluated after the algorithm performed the specific iteration, but before the next iteration starts. If the sentry particle's fitness value differs after re-evaluation with more than a specified value, the swarm is notified that a change in the environment has occurred. Once a change has been detected, one of the following approaches are used to react to the change:

- The pbest of the particle is set to its current position if the current position dominates the pbest, referred to as response$_a$.
- The pbest of the particle is set to its current position, referred to as response$_b$.

The performance of the modified MOPSO algorithm using each of the two response approaches is measured using GD and the variance of GD in the objective space. Two

DMOOPs were used to test the modified MOPSO, namely FDA1 and a modified version of FDA2. When solving FDA1, for $n_t = 10$ and $\tau_t = 10$ response$_b$ obtained the best average GD values with a slightly higher standard deviation of GD values. However, for $n_t = 10$ and $\tau_t = 50$, the performance of both approaches were very similar. When solving the modified FDA2 with $n_t = 10$ and $\tau_t = 10$, response$_a$ obtained a better average GD value and a better standard deviation GD value. However, for $n_t = 10$ and $\tau_t = 50$, response$_b$ obtained a better average GD value and a better standard deviation GD value. Therefore, in fast changing environments response$_a$ outperformed response$_b$, but in slower changing environments both approaches performed well. The modified MOPSO algorithm using response$_b$ was compared against the original NSGA-II algorithm that was not adapted for DMOO. FDA1 and the modified FDA2 DMOOPs were used with $n_t = 10$ and $\tau_t = 50$. The results indicate that the modified MOPSO algorithm outperformed NSGA-II on both functions and that there was a statistical significant difference in the GD values with a confidence level of 5% using t-tests [102].

The next PSO algorithm proposed for DMOO was the maximinPSO algorithm [107] extended by Li *et al.* to solve DMOOPs. MaximinPSO evaluates the fitness of an individual with a maximin fitness function that takes into account non-dominance and diversity [107]. When solving DMOOPs, maximinPSO does not test for changes in the environment. At each iteration the particle's pbest is set to its current position. Therefore, only the gbest influences the particle's velocity and new position, i.e. the social-only PSO model is used. According to the empirical results of Kennedy [93], the social-only PSO is more efficient than both the full PSO that uses the pbest and gbest to update the particle's velocity and the cognitive-only PSO that uses only the pbest to update the particle's velocity. The social-only PSO exploits more and therefore converges quicker. However, the social-only PSO is susceptible to local optima [93].

The adapted maximinPSO was tested against the DMOOPs introduced by Jin *et al.* [90]. HVR and IGD were used to measure the performance of the algorithm. The adapted maximinPSO was compared against the original maximinPSO that does not reset the particle's pbest to its current position. The results indicate that for fast changing environments the adapted maximinPSO outperformed the original maximinPSO algorithm.

8.1.3 Converting Dynamic Multi-objective Optimisation Problems into Single Multi-objective Optimisation Problems

Another approach to solve DMOOPs is to convert the DMOOP into various SMOOPs. Wang and Dang [153] proposed a new approach where the time period of the DMOOP is divided into several smaller time intervals. For each time interval a SOOPs is defined by limiting the DMOOP to the specific time interval. Each SMOOP is then transformed into a new two-objective optimisation problem where one objective is to increase the diversity of the solutions and the other objective is to increase the quality of the found non-dominanted solutions. Uniform cross-over is used to avoid cross-over between two individuals that are in close proximity to each other during the first few iterations of the algorithm run. The proposed EA was compared against NSGA-II using FDA1, FDA2 and FDA3. The algorithms' performance was measured using the C-metric and the U-measure. The results indicate that the proposed algorithm successfully tracked the changing POF over time. Furthermore, the proposed algorithm outperformed NSGA-II with regards to both performance measures.

A memetic algorithm (MA) [119] that incorporates a sequential quadradic programming (SQP) solver was proposed by Isaacs *et al.* [87, 88]. Change detection is done by evaluating randomly selected individuals, and if their fitness has changed it indicates a change in the environment. Where EAs use genetic operators (such as cross-over and mutation) to evolve the individuals, the MA uses SQP. The population is randomly initialised within the search space of the DMOOP. The extreme solutions of the POF is calculated by minimising each objective function seperately as a SOOP. Then the range of each objective is sub-divided into small intervals of a specified size. For each of these intervals, a SOOP is solved, where the objective's range is within the smaller interval. This process of solving the DMOOP through a series of SOOPs are referred to as an orthogonal epsilon-constrained formulation of the DMOOP. For each of the SOOPs, an individual of the population is randomly selected as the starting point. The MA was tested against FDA1 and a modified version of FDA2. As expected, an increase in the number of SQP iterations resulted in the found solutions being closer to the POF. However, increasing the number of SQP iterations requires more function evaluations [87]. The MA was also applied to train a neural network for an unmanned aerial vehicle problem. The results indicate that the MA trained the neural network faster than the online Levenberg-Marquardt algorithm [87].

In another study [88], the MA was compared with an EA that incorporates simulated binary cross-over (SBX) and polonomial mutation. When a change in the environment occurs, the EA re-starts, i.e. all solutions in the population are replaced by new solutions with randomly created positions in the search space. MA and the EA were also tested against FDA1 and a modified version of FDA2. The results indicate that MA obtained a better accuracy than the EA for both functions. The MA framework of Isaacs *et al.* was extended by Ray *et al.* [127]. In the extended version, the MA framework is extended so that either SQP or a sub-EA is used. The sub-EA evolves the population of the MA and uses SBX and polonomial mutation. The performance of the MA and sub-EA versions were compared using FDA1 and a modified version of FDA2, with $n_t = 10$ and $\tau_t = 5$. The results indicate that MA consistently obtained a better accuracy (measured using GD in objective space) than the sub-EA version [127].

Liu and Wang [111, 112] proposed a modified MOEA to solve DMOOPs, referred to as DMEA. The total time of the DMOOP is divided into smaller time intervals. For each time interval the DMOOP is considered as a static MOOP. For each of the MOOPs, an EA is used to optimise the problem. If a change in the environment occurs, the algorithm re-starts by creating a new initial population.

DMEA was tested against two DMOOPs. No performance measures were used, but it did seem as though DMEA was tracking the changing POF according to the POF plots [111]. In another study, DMEA was tested against four DMOOPs. Once again, no performance measures were used and only graphs of the approximated POF were provided. Even though the authors claim that, according to the POF plots DMEA successfully solved the DMOOPs, this is not the case. Closer inspection of the POF plots for DMOOP G3 (FDA2 of Farina *et al.*) reveals that the algorithm lost track of the changing POF.

8.1.4 Generic Extensions for Dynamic Multi-objective Optimisation Algorithms

Various contributions were made to improve dynamic MOEA (DMOEA) algorithms. These contributions are more generic in nature and can therefore easily be incorporated into various DMOO algorithms. Guan *et al.* [74] proposed an inheritance strategy for MOEAs solving DMOOPs. When a change in the environment occurs, the MOEA selects good performing individuals according to the new objectives, and then optimises only the selected individuals until the next change occurs. The inheritance strategy was incoporated into three MOEAs, namely PAES [98], SPEA [172] and NSGA-II [47, 42]. After a change in the environment occurs, PAES re-evaluates the solutions in the archive with regards to the new objectives, the dominated solutions are removed, and only the non-dominated solutions survive. Similar to PAES, SPEA re-evaluates the solutions in the external population and only the non-dominated solutions survive. However, since NSGA-II does not use an archive or an external population to store non-dominated solutions, the whole population is re-evaluated after a change occurs. The whole population is then re-sorted based on Pareto-ranking according to the new objectives. To evaluate the influence of the inheritance strategy on the performance of the MOEAs, the performance of each MOEA without the inheritance strategy was compared against the same MOEA incorporating the inheritance strategy [74]. The MOEAs were tested against three DMOOPs where objective replacement was used to adapt the MOOPs to DMOOPs. The study revealed that the MOEAs with the inheritance strategy found more non-dominated solutions than the MOEAs without an inheritance strategy. MOEAs with the inheritance strategy also converged closer to the true POF than MOEAs without the inheritance strategy. Furthermore, MOEAs with the inheritance strategy found solutions with either a similar or better distribution than MOEAs without the inheritance strategy.

Four re-initialisation strategies that can be applied to any DMOEA when a change in the environment is detected, was proposed by Zhou *et al.* [166]. The four strategies are:

- Randomly re-initialise all new solutions within the bounds of the search space (RND).

- Add Gaussian noise, based on the last time window, to each individual (VAR).
- Sample solutions around predicted locations of the POS (PRE). The last two time windows are used to predict the next location of the POS.
- Create half of the solutions using PRE and the other half using VAR (V&P).

The effectiveness of these four strategies were evaluated on FDA1 and ZJZ. The algorithms' performance were evaluated using GD in decision space and HVD. The results indicate that RND did not perform well, since all previous knowledge were removed by randomly re-initialising all individuals. For FDA1, PRE obtained the best performance independent of the time window size, V&P performed well, VAR performed poorly and RND performed really bad. For ZJZ, the results depended on the time window size. With a time window size of 500, VAR and V&P performed similarly. However, when the time window size increased, V&P and PRE outperformed the other approaches.

When an algorithm solves DMOOPs, one of the issues that should be addressed is how to re-use past knowledge of the found POS when the environment changes. Wang and Li [155] proposed four memory-based approaches, namely:

- a restart scheme, where all individuals are re-initialised to random positions within the search space.
- an explicit memory scheme, where each individual is replaced by either a randomly created solution within the search space or by re-evaluating a randomly selected solution from the archive.
- a local search memory scheme, where each individual is replaced by either a randomly created solution within the search space or by re-evaluating a randomly selected solution from the archive and then performing a local search on the re-evaluated solution.
- a hybrid memory scheme, where each individual is replaced by either a randomly created solution within the search space, or by either re-evaluating a randomly selected solution from the archive or by re-evaluating a randomly selected solution from the archive and then performing a local search on the re-evaluated solution.

These memory schemes were incorporated into a modified NSGA-II algorithm (INSGA-II) that uses an archive. A GA-DE operator is introduced to create offspring, utilising the fast convergence of GAs and DEs' ability to find diverse solutions.

The various approaches were evaluated using FDA1, the DMZDT functions and WYL. The algorithms' performance were measured using IGD in objective space. The results indicate that the explicit, local search and hybrid memory schemes improved INSGA-II's performance. However, as the frequency of changes increased, the effectiveness of the memory schemes decreased. The local search memory scheme was more robust than the other memory schemes. Furthermore, the explicit memory scheme experienced a loss in diversity when the change frequency decreased.

With EAs, immigration schemes enable insertion of new information into the existing genetic pool of the population and therefore increases the diversity of the population. Azevedo and Araújo [3] proposed a new immigration scheme for EAs to solve DMOOPs, called the generalised immigrants-based diversity generator (gIDG). The immigration scheme was incorporated into NSGA-II [47, 42] to investigate whether the new immigration scheme leads to better performance when solving DMOOPs. gIDG is incoporated into NSGA-II in the following way: a specified number of worst individuals, n_w, are replaced with generated immigrants after cross-over and mutation, but before selection of the population for the next generation. This ensures that the immigrants do not influence the generation of offspring, but enable the immigrants to compete with the newly generated offspring for survival. The initial results indicate that a higher quality of front one solutions (non-dominated solutions with Pareto rank one) was maintained over time if the immigrants that were introduced consisted of a mixture of elite-based and random immigrants [3]. Elite-based immigrants are generated by first sorting the population according to dominance and then selecting the first n_w individuals from the sorted population to create an elite population. Individuals of the elite population are then randomly selected and mutated to create the elite-based immigrants. Random-based immigrants are generated by randomly sampling n_w solutions from a probability distribution function. Experiments were conducted using FDA1, FDA2 and a modified version of dMOP3. The algorithm's performance was measured against the offline HV. The results indicated that NSGA-II with gIDG obtained statistically significantly better HV values than NSGA-II without any immigration scheme, NSGA-II with only random-based immigrants, and NSGA-II with only elite-based immigrants [3].

An important aspect when solving DMOOPs is deciding which solution to use. Deb *et* *al.* [46] highlighted the need for automated decision making when solving DMOOPs, since a solution has to be selected and implemented before an environment change occurs. Deb *et al.* used a utility measure to automate the decision making process. Another approach is to enable a decision maker to interactively and easily indicate his/her preference [113]. Therefore, Liu *et al.* [113] introduced an artificial immune system algorithm, called the sphere-dominance preference immune-inspired algorithm (SPIA). SPIA introduces a new concept called sphere-dominance that enables a decision maker to interactively define his/her preferences with regards to the solutions of the DMOOP. A sphere (or multiple spheres) with a reference point and a specified radius is defined by the decision maker. If a solution's distance to the reference point is less than the specified radius, it belongs to the sphere. Sphere-dominance then replaces the normal Pareto-dominance relation and is defined as follows:

- a solution x_a sphere-dominates a solution x_b if x_a is a member and x_b is not a member of the sphere respectively.
- a solution x_a sphere-dominates a solution x_b if both solutions are members of the sphere and x_a Pareto-dominates x_b.

This new dominance relation drives the search towards the decision maker's desired area of the found POF. Two hypermutation methods are used, namely Gaussian hypermutation and predictive hypermutation. With predictive hypermutation, a time series is developed for each antibody in the population and the new antibodies are predicted using a linear model. A progress rate that measures the improvement in the antibody quality is calculated. Then, based on the progress rate, one of the hypermuation methods are selected. The performance of SPIA was evaluated using FDA3 and FDA4 with $n_t = 10$ and $\tau_t = 50$. The convergence of SPIA was measured with GD in the objective space. For FDA3, SPIA performed well with regards to GD when searching for the whole POF, one preference region in the middle of the POF, two preference regions at the edges of the POF, and two preference regions away from the edges of the POF. When the decision maker defined two preference regions away from the edges of the POF, SPIA struggled during the first 200 generations to converge towards these regions, but then converged very well between generations 200-500. For FDA4, SPIA performed well with regards to GD when searching for the whole POF and one preference region. However, with two preference regions SPIA did sometimes struggle to find many solutions and to converge towards the two preference regions. However, since the spread of solutions of FDA5 changes over time, finding solutions in two preference areas of the POF is a difficult task. One problem with the sphere preference approach is that the reference point has to change as the POF changes over time. Therefore, the algorithm enables the decision maker to interactively change his/her preference during the optimisation process.

Many real-world problems are computationally expensive to solve. However, when solving DMOOPs, parallel processing may speed up the execution time of the algorithm. Therefore, Cámara *et al.* [17] introduced a parallel algorithm approach that can be applied to any MOEA. The parallel algorithm has a master process that subdivides the population into sub-populations of equal size and then sends a sub-population to each worker process. Every worker process optimises the MOOP for a fixed number of iterations in the worker process's assigned search space using the assigned MOEA and keeps only the non-dominated solutions. After a fixed number of iterations, each worker process sends the found non-dominated solutions to the master process. The master process adds all non-dominated solutions obtained from the worker processes into a new population and then runs a MOEA on the new population for a specified number of iterations. After the specified number of iterations the master subdivides the population into sub-populations and the whole process is then repeated until the stopping condition has been reached. Cámara *et al.* compared the speedup obtained by the parallel approach by applying the approach to four MOEAs, namely the single front genetic algorithm (SFGA) [20], SFGA2 [20], SPEA2 [170] and NSGA-II [47]. The algorithms were tested against FDA1, FDA2$_{Camara}$ and FDA3$_{Camara}$. The algorithms' performance were measured against the HV, the maximum HV, *acc*, *stab* and *reac*. Using more than one worker processes lead to better performance by the algorithms. Furthermore, the results indicate a small computational time decrease for SFGA and SFGA2, and a huge decrease in computational time for NSGA-II and SPEA2.

8.1.5 Prediction-based Approaches

When solving real-world MOOPs and DMOOPs, function evaluations can be computationally expensive. Therefore, techniques were developed that use prediction to decrease the number of function evaluations without decreasing the quality of found solutions [145]. Hatzakis and Wallace [76] introduced a hybrid algorithm that incorporates a forecasting technique with an evolutionary algorithm, referred to as the dynamic queuing multi-objective optimizer (D-QMOO). D-QMOO is an adaptation of the queuing multi-objective optimizer (QMOO) algorithm proposed by Leyland [104] to solve MOOPs. D-QMOO is a steady-state clustering MOEA. Each cluster's population consists of two sub-populations, referred to as the *front* and the *cruft*. The front consists of only non-dominated solutions of Pareto-rank one and the cruft only has dominated solutions of Pareto-rank greater than one. The task of the individuals in the front is to converge to the current true POF. In contrast, the goal of the individuals in the cruft is to increase diversity to enable the algorithm to search and discover a new optimum. If the front has reached its maximum size, individuals are eliminated in such a way that the maximum HV is preserved. D-QMOO preserves elitism by only adding non-dominated individuals to the front. Any dominated individual can be added to the cruft and if the cruft's maximum size has been reached, individuals are removed. Individuals are eliminated from the cruft based on either their age (where the oldest individuals are removed) or crowding in the decision space (individuals in more crowded regions in the decision space are removed).

If D-QMOO incorporates forecasting, a third sub-population (referred to as the *prediction set*) is added when a change in the environment occurs. Forecasting is done using stochastic time series models. The input to the forecasting model is a time series of a specified number of selected points on the POF during the previous time steps. Individuals are then placed on the predicted location of each anchor point and these individuals are then added to the prediction set. The individuals of the prediction set are absorbed by either the front or the cuft during the next step of the optimisation process according to their fitness. Therefore, the prediction set only exists directly after a change in the environment occurs. If the forecasting is accurate, individuals of the prediction set are placed in close vacinity of the new optima, leading to faster convergence towards the new POF. However, the accuracy of the forecasting depends on the accuracy of D-QMOO. If D-QMOO did not converge properly to the true POF in the previous time-steps, the input into the forecasting model will lead to errors in the prediction of the location of the anchor points. Furthermore, even if the input into the forecasting model is correct, the predicted location of the anchor points may be inaccurate. Therefore, only a small number of individuals should be added to the prediction set. D-QMOO was evaluated on FDA1. The results indicate that the feed-forward prediction strategy lead to better convergence when the environment changed frequently, but did not really improve the performance when the change frequency was low [76].

DMEA [111, 112] is extended by Liu to incorporate an estimation of the next generation's POS [110] to develop the CDDMEA algorithm. The new POS is estimated by calculating the core of the POS at various time steps. The core of a POS is the average solution of the POS, i.e. the mean is calculated for each dimension of the solutions. The difference between the core solutions at time $t-1$ and time $t-2$ is then added to a solution at time t to estimate the solution at time $t+1$. CDDMEA was compared against DNSGA-II-A [46] using a DMOOP of Jin and Sendhoff and FDA2. The performance of the algorithms were measured using the U-measure. The results indicate that CDDMEA outperformed DNSGA-II on both DMOOPs. It should be noted that this prediction approach is based on the POS. Therefore, errors in previously found POSs may cause the algorithm to lose track of the changing POF or POS.

Talukder *et al.* [145] introduced a variation operator for MOEAs and Talukder and Kirley [96] extended the variation operator for DMOEAs. The variation operator approximates the next POF and then inversely maps the POS from the approximated POF using integral transformation [96]. This mapping procedure replaces the standard cross-over and mutation used in MOEAs. Fourier transformation is used as the integral transformation and therefore the dimensions of the input (objective function values) and output values (decision variable values) should be the same. However, with DMOOPs the dimension of the decision variables is usually not the same as the number of objective functions. In order to overcome this problem, each of the design variables and objective function values are considered as seperate input/output values. The variation operator is incorporated into the NSGA-II algorithm after non-dominated sorting to create new individuals from the next predicted POF. The newly created individuals are then combined with the parent population and the remainder of the NSGA-II algorithm steps are performed as usual. The modified NSGA-II algorithm was compared against DNSGA-

II-A [46] using FDA2 and modified versions of FDA3 and FDA5. The performance of the algorithms were measured using the HVR, but with the POF found by DNSGA-II-A used as the approximated front and the POF found by Talukder and Kirley's modified NSGA-II (NSGA-II-TK) used as the known POF. Therefore, an HVR value at time t greater than one indicates that at time t NSGA-II-TK performed better than DNSGA-II-A and vice versa. For FDA2, NSGA-II-TK outperformed DNSGA-II-A between generations 150 and 250, and for FDA3 NSGA-II-TK outperformed DNSGA-II-A for most generations. For FDA5, NSGA-II-TK outperformed DNSGA-II-A for all generations. However, it should be noted that the value with which the POF values decrease between various predicted POFs is problem dependent. Furthermore, the variation operator can only be used for Type II and Type III DMOOPs where the POF changes over time. The variation operator also requires that the population is sorted into non-dominated fronts (Pareto-ranking) and that there is more than one front. Therefore, the variation operator cannot be used by algorithms that store solutions in an archive without ranking solutions in various fronts. An advantage of this predictive approach is that the objective functions do not have to be differentiable.

D-NSGA-II was adapted by Roy and Mehnen [129]. After a change in the environment occurs, the parent population is discarded, only the offspring is re-evaluated and the algorithm is re-started. With this approach, no new individuals are introduced as is the case with DNSGA-II-A and DNSGA-II-B. Furthermore, forecasting or prediction is incorporated into the algorithm by dividing the objective space into a grid of hyper-cubes where each cube represents a section of the POF for a specific time t. At each time t, the means of the coordinates of the points within the cube is calculated to determine representative points. Each representative point in the grid is assigned a two dimensional time series. Then for each objective, a state space model is selected to model the objective's multi-variate time series. The DNSGA-II with forecasting uses the k forecasted values for k iterations after every pre-defined number of iterations. During the k iterations no function evaluations are performed. After the k iterations the DNSGA-II algorithm proceeds in its normal way. Furthermore, the objective functions are transformed according to defined desirability functions to guide the MOEA towards certain sections of the POF that experts with the required knowledge expect to be of higher relevance than other parts of the POF. The algorithm was tested on a real-world problem where every 40 generations forecasted values are used for 5 iterations. The results indicated that the multivariate analysis of more than four time series at a time resulted in forecasts with poor confidence intervals. Furthermore, the algorithm struggled when there was missing data, for example when the POF was disconnected or the POF had a low density of solutions in certain areas. According to Roy and Mehnen [129] this problem can be overcome by using larger population sizes and shorter intervals between the forecasting periods.

Recently Koo *et al.* [100] adapted MO-EGS [65] for DMOO and referred to the new algorithm as dMO-EGS. An optional prediction strategy can be incorporated into dMO-EGS, which is then referred to as dynamic MO-EGS with prediction gradient (dMO-EGS-PG). If the prediction strategy is incorporated, an archive centroid is used to predict the movement of the POS.

The gradient prediction strategy relies on a POS that changes in a similar way over time. Therefore, if the POS changes randomly, the gradient predition strategy will be of little or no use. Furthermore, since the prediction strategy relies on the previous values of the POS found by the algorithm, the accuracy of the gradient relies on the accuracy of the algorithm during the previous time steps. dMO-EGS also incorporates storing past information in addition to an archive.

In order to determine the effect of the prediction gradient strategy on the performance of dMO-EGS, dMO-EGS and dMO-EGS-PG were compared using FDA1, FDA3, DIMP1 and DIMP2. The results indicate that dMO-EGS-PG outperformed dMO-EGS on all four DMOOPs with regards to both convergence to the true POS and the spread of the solutions. Furthermore, dMO-EGS-PG converged to the true POS of DIMP2 in situations when dMO-EGS failed to converge, i.e. even when there were severe changes to the environment or the frequency of changes was high [100].

8.2 Summary

This chapter discussed population-based algorithms proposed for DMOO. Five main categories of DMOO algorithms were identified, namely adapted SMOO algorithms, new types of CI algorithms, conversion of DMOOP into multiple SMOOPs, generic extensions for DMOO and prediction-based approaches.

Many extensions were proposed to SMOO algorithms to solve DMOOPs. Most extensions were proposed to EAs. However, PSO and AIS were also adapted for DMOO. Three new types of CI algorithms were proposed to solve DMOOPs, namely ALife, membrane computing and a competitive-cooperative EA.

Generic extensions or strategies were also proposed that can be applied to any DMOO algorithm. These extensions included an inheritance strategy, re-initialisation strategies, memory-based approaches, immigration schemes, the concept of sphere-dominance, and parallel processing.

In order to reduce the number of function evaluations without decreasing the quality of found solutions, prediction based approaches were introduced. These approaches use knowledge of previous environments to predict the new location of either the POS or POF.

Even though many EA algorithms were proposed for DMOO, only two PSO-based algorithms were introduced to solve DMOOPs. The next part of the thesis discusses a new multi-swarm PSO algorithm for DMOO, namely DVEPSO. The next chapter introduces the extensions made to VEPSO for DMOO and investigates the effect of various guide update approaches on the extended algorithm, i.e. DVEPSO.