4 Memetic Self-Adaptive Firefly Algorithm

Iztok Fister[1], *Xin-She Yang*[2], *Janez Brest*[1] *and Iztok Jr. Fister*[1]

[1]Faculty of Electrical Engineering and Computer Science, University of Maribor, Maribor, Slovenia; [2]Department of Design Engineering and Mathematics, School of Science and Technology, Middlesex University, The Burroughs, London, UK

4.1 Introduction

Nature has always been a source of inspiration for scientists when searching for solutions to given problems. For example, the collective behavior of social insects like ants, termites, bees, and wasp, as well as other animal societies like flocks of birds or schools of fish, has inspired scientists to design intelligent multiagent systems (Blum and Li, 2008). In the natural world, colonies of social insects consist of a huge number of unsophisticated beings/agents (i.e., individuals) so that a colony as a whole can accomplish complex tasks in cooperation. These tasks are coordinated without any centralized control and are thus self-organized. The fundamentals of collective behavior regarding individuals in colonies have inspired scientists to solve some complex, practical problems. The research field that exploits swarm behavior is referred to as swarm intelligence. Swarm intelligence can be emergent in complex systems, especially, in those systems that are initially demand flexibility and robustness.

The term swarm intelligence was first used by Beni and Wang (1989) in the context of a cellular robotic system. Nowadays, this term also extended to the field of optimization, where the techniques based on swarm intelligence have been applied successfully. Examples of notable swarm intelligence optimization techniques are ant colony optimization (Dorigo and Di Caro, 1999; Korošec et al., 2012), particle swarm optimization (Kennedy and Eberhart, 1999), and artificial bee colony (ABC) (Karaboga and Basturk, 2007). Today, the most promising swarm intelligence optimization techniques include the firefly algorithm (FFA) (Gandomi et al., 2011, 2013; Yang, 2008; Yang et al., 2012; Talatahari et al., 2012), the cuckoo search (Yang and Deb, 2009), and the bat algorithm (Yang, 2010).

Swarm intelligence refers to the collective behavior of social individuals. In nature-inspired metaheuristic algorithms, these individuals are represented as a population of solutions. The population of solutions is maintained, instead of

Swarm Intelligence and Bio-Inspired Computation. DOI: http://dx.doi.org/10.1016/B978-0-12-405163-8.00004-1

searching for a single solution to a problem of interest. Therefore, this type of algorithms is also called population-based. From individuals in a population (also parents), population-based algorithms, such as evolutionary algorithms (EAs), are able to produce new solutions (e.g., offspring) in terms of the operations of mutation and crossover (Eiben and Smit, 2003).

However, population-based algorithms have many advantages over single-point search algorithms, as summarized as the following five points (Prügel-Bennett, 2010):

1. Building blocks are put together from different solutions through crossover.
2. Focusing search again relies on the crossover and means that if both parents share the same value of a variable, then the offspring will also have the same value of this variable.
3. Low-pass filtering ignores distractions within the landscape.
4. Hedging against bad luck in the initial positions or decisions it makes.
5. Parameter tuning is the algorithms' opportunity to learn good parameter values in order to balance exploration against exploitation.

The exploration and exploitation in the FFA need to be defined before taking a closer look at these components for the search process. Exploration and exploitation are the two cornerstones of problem solving by iterative search (Črepinšek et al., 2011). Exploration refers to the moves for discovering the entirely new regions of a search space, while exploitation refers to the moves that focus its search on the vicinity of promising, known solutions found during the search process. Other terminologies for exploration and exploitation used by Blum and Roli (2003) are diversification and intensification that refer to medium- to long-term strategies based on the usage of memory, while exploration and exploitation refer to short-term strategies tied to randomness. In this chapter, the terms exploration and exploitation are used for consistency.

Exploration and exploitation must be balanced in order to make a search move efficiently and effectively. Namely, too much exploration can lead to inefficient search, while too much exploitation can cause the premature convergence of a search algorithm where the search process, usually due to reducing the population diversity, can be trapped into a local optimum (Eiben and Smit, 2003).

How to balance these two components of the search? To date, the balance between exploration and exploitation has been managed indirectly, i.e., by proper control parameter settings. For example, EAs involve various variations (e.g., crossover and mutation) and selection operators that can be controlled via control parameters. A suitable parameter setting depends on the given problem. In fact, a parameter setting suitable for a certain problem can be unsuitable for another. Furthermore, set parameters that are suitable at the beginning of a search process can become unsuitable during the maturing phases of the search. Therefore, a necessity has been arisen for adapting parameters during a search process.

The most complete taxonomy of the parameter setting can be found by Eiben et al. (1999). According to this taxonomy, the authors distinguished between two major forms of parameter setting: parameter tuning and parameter control. In the former case, good values of parameters are set before the run. These values then

remain unchanged throughout the run. In the latter case, the parameters start with initial values that are then changed during the run. These values can be changed: deterministic, adaptive, or self-adaptive. Deterministic parameter control occurs when the values of parameters are changed by some deterministic rule. A characteristic of adaptive parameter control is that the direction or magnitude of the change depends on feedback from the search. One example of this control is represented by Rechenberg's 1/5 success rule (Rechenberg, 1973). Finally, the parameters are encoded into representations of individuals and undergo variation operators by self-adaptive parameter control.

The proposed FFA tries to balance the exploration and exploitation more directly during the run of a search process. Directly controlling the balance is difficult (Črepinšek et al., 2011). The first question that must therefore be answered is how to measure exploration and exploitation during a search. Typically, exploration and exploitation are implicitly measured using a diversity of population that plays a crucial role in the success of optimization. Diversity refers to differences among individuals. These differences can arise either at the genotype or phenotype levels. The genotype refers to the structural characteristics of individuals' representation. Phenotype determines the quality of an individual. Population diversity measures how similar individuals are to each other (Neri, 2012). When individuals are distributed over the whole search space, the population has high diversity. On the other hand, when individuals are crowded to a certain region of the search space, it has low diversity.

Unfortunately, diversity is only roughly related to exploration and exploitation (Črepinšek et al., 2011). High diversity is not necessarily a sign that the population consists of fit individuals. It only indicates that individuals are exceedingly dissimilar. On the other hand, low diversity can indicate that the search algorithm has converged to the some optimum. This optimality can either be global or local. In the former case, the algorithm finds the optimal solution, while in the latter case it is trapped into a local optimum, e.g., in case of an EA, premature convergence has arisen.

In place of premature convergence, a phenomenon of stagnation can be typical for swarm intelligence, which can occur when the search algorithm cannot improve the best performance (also fitness) although the diversity is still high. Thus, less-promising regions of the search space are explored.

First of foremost, swarm intelligence is concerned in optimization and robotics. This chapter, however, is devoted to optimization, i.e., how to solve optimization problems using swarm intelligence techniques. One of the latest swarm intelligence techniques is the FFA that is the main subject of this chapter. The main characteristic of fireflies is their flashing light that can be visible, especially, on summer nights in tropical and temperature regions. Such flashing light can be expressed in the form of a physical equation regarding light intensity. This equation can be associated with the objective function of the problem to be optimized. The main advantage of the FFA is to search for more optima simultaneously and thus, it is more suitable for nonlinear, multimodal problems.

In the proposed FFA, a stagnation phenomenon is signaled when the diversity of the population being measured on a phenotype level remains stable, and the best

fitness does not improve regarding the prescribed number of generations. If the stagnation condition is detected, the proposed FFA recruits individuals with higher diversity. Thus, the diversity of the entire population is increased and it can therefore be supposed that the search algorithm needs to focus itself on exploring a broader region of the search space.

Three additional features have been applied to the proposed FFA as follows:

1. self-adaptation of control parameters,
2. a new population model,
3. local search heuristics

that will be explained in the remainder of this chapter. A global search algorithm hybridized with a local search is referred to as memetic algorithm by Moscato (1999). In our case, the memetic self-adaptive FFA (MSA-FFA) has been developed and applied to a graph 3-coloring problem (3-GCP). The extensive experiments to be given below show that the results of MSA-FFA are comparable with the results of other tested algorithms.

The rest of this chapter is organized as follows. In Section 4.2, optimization problems and their complexity are discussed. The principles of the MSA-FFA are explained in Section 4.3. Section 4.4 describes a case study in which the MSA-FFA is applied to a graph 3-coloring. Besides a detailed description of this algorithm, the extensive experimental work is illustrated and the obtained results are compared with three other well-known graph coloring algorithms, i.e., Tabucol, HEA, and EA-SAW. Finally, Section 4.5 summarizes the results and a closer look at future work.

4.2 Optimization Problems and Their Complexity

From a system analysis point of view, an optimization problem can be seen as a system consisting of three components: input data, output data, and the model (Eiben and Smit, 2003). The model is treated as a black box that transforms input data into output data. Knowing the model and output data, the optimization problem becomes how to find input data that satisfies the criterion of optimality (Figure 4.1).

Let us suppose, a traveling salesman problem (TSP) is given. Here, an equation (model) is provided that calculates their length (output data) for each arbitrary cycle (input data). The length of the cycle is calculated by an objective function f_{obj}. The task of the optimization system is to find the cycle with the shortest length

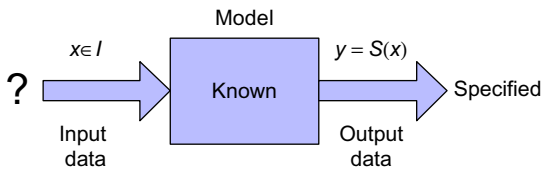

Figure 4.1 Optimization problem.

(goal). The shortest cycle represents an optimal solution that can be written as $\mathbf{s}^* = S^*(\mathbf{x}) = S(\mathbf{x}^*)$, while its optimal value can be expressed as $f^*_{\text{obj}}(\mathbf{s})$. Note that only one set of input data $\mathbf{x}$ can be put on the input. This set of data is referred to as an instance I. Normally the instance $\mathbf{x} \in I$ is a vector of elements x_i that are called design or decision variables. In line with this, the optimization problem can be formally defined as a quadrille $P = (I, S, f_{\text{obj}}, g)$, where

- I denotes all instances of the input data;
- S is the function that to each instance $\mathbf{x} \in I$ assigns a set of feasible solutions $S(\mathbf{x})$;
- f_{obj} is the objective function that to each feasible solution $\mathbf{s} \in S(\mathbf{x})$ of instance $\mathbf{x} \in I$ assigns the value $f_{\text{obj}}(\mathbf{s}) \in \mathbb{R}$;
- g denotes the goal that determines when the minimum ($g = \min$) or maximum ($g = \max$) value of objective function is searched for.

Usually, the fitness function f is used in place of the objective function f_{obj}. If we suppose that $\max(f_{\text{obj}}(\mathbf{s})) = \min(-f_{\text{obj}}(\mathbf{s}))$, then the goal $g = \max$ can always be transformed into $g = \min$. In other words, we always search the minimum value of the transformed objective function. As a result, the optimization problem can be reduced to a triple $P = (I, S, f)$. It can arise in three forms, as follows:

1. *Construction form*: The optimal solution $\mathbf{s}^*$ and to it the belonging value of the fitness function $f^*(\mathbf{s})$ need to be calculated for the instance $\mathbf{x} \in I$.
2. *Nonconstruction form*: The optimal value of the fitness function $f^*(\mathbf{s})$ is needed for the instance $\mathbf{x} \in I$.
3. *Decision form*: For the particular instance $\mathbf{x} \in I$, it should be discovered whether the optimal value of fitness function $f^*(\mathbf{s})$ is better than a certain prescribed constant K, more formally if $f^*(\mathbf{s}) \leq K$.

For example, the shortest cycle (a sequence of cities) and its length need to be found when the TSP problem is given in its construction form. The length of the shortest cycle is demanded when the TSP problem is in nonconstruction form, while in the decision form of TSP, it is necessary to answer the question whether the shortest cycle found is shorter than a certain prescribed length K.

How hard to solve the problem is, depends not on the problem solver, but rather on the features of the fitness function within the search space. The visualization of fitness values' distribution forms a kind of fitness landscape introduced by Wright (1932). Mathematically, the fitness landscape is defined as (S, f, d), where S denotes the search space, f the fitness function that assigns to each solution $\mathbf{s} \in S(\mathbf{x})$ with the fitness value $f(\mathbf{s})$, and $d: S \times S \to \mathbb{R}$ is a distance metric that defines the spatial structure of the landscape (Stadler, 1995). The fitness landscape reflects the following characteristics of the problem: multimodality, separability, noise, time dependency, etc.

Optimization problems can be further divided into continuous and discrete. The latter is also referred to as combinatorial. The decision variables for a continuous optimization problem can occupy values within the domain of real values $\mathbb{R}$, while the decision variables for a combinatorial problem have discrete values.

According to the number of objectives involved in the optimization problem, this can be divided into single-objective and multiobjective (also multicriteria).

The task of single-objective optimization is to find the optimal solution according to only one objective function. When the optimization problem involves more than one objective function, the task is to find one or more optimal solutions regarding each objective (Deb, 2001). Here, a solution that is good with respect to one objective can be worse for another, and vice versa. Therefore, the goal of multiobjective optimization is to find a set of solutions that are optimal with respect to all other objectives, and such a set of solutions form a so-called Pareto front. Interestingly, most real-world problems are multiobjective.

Many practical problems have complex constraints. Namely, not all possible combinations of decision variables represent valid solutions. Constraint problems can be divided into two different types: constraint satisfaction problems (CSPs) and constraint optimization problems (COPs). In contrast, if the problem is unconstrained, it is referred to as a free optimization problem (Eiben and Smit, 2003). CSP is defined as a pair $\langle S, \phi \rangle$, where S denotes a search space and ϕ is a Boolean function on S that represents a feasibility condition. In fact, this function divides the search space S into feasible and unfeasible regions. A solution of the CSP is each $\mathbf{s} \in S$ with $\phi(\mathbf{s}) = \text{true}$. On the other hand, COP is defined as a triple $\langle S, f, \phi \rangle$, where S denotes a search space, f is a real-valued fitness function, and ϕ is a Boolean function on S. A solution of this problem is that $\mathbf{s} \in S(\mathbf{x})$ with $\phi(\mathbf{s}) = \text{true}$ and $S(\mathbf{x}) = S(\mathbf{x}^*)$.

In general, algorithms are step-by-step procedures for solving problems (Garey and Johnson, 1979). These procedures can be computer programs, written in specific programming languages. The first programs tried to solve optimization problems by exhaustive search, i.e., by enumerating all possible combinations of decision variables. However, these programs are too time-consuming and impractical for solving most real-world problems. Therefore, more approaches were developed that tried to solve these problems approximately in a reasonable time. Nowadays, there exist optimization algorithms that search for solutions by using gradient-based and heuristic-based search techniques (Deb, 2001). Deterministic and stochastic search principles are applied in these algorithms. While the deterministic principle behaves in a predictably mechanical way, because it always produces the same results, the stochastic principle involves a randomness step within the algorithm. The necessity of widening the applicability of optimization algorithms to various new problem domains has led to robust optimization algorithms inspired by natural and physical principles or characteristics. Essentially, EAs and swarm intelligence belong to this class of algorithms, which also belong to metaheuristic algorithms (Yang, 2008).

If an algorithm successfully solves all instances of some problem P, then we can say that it is capable of solving the problem P. Usually, we are interested in which algorithm solves the problem more efficiently. Normally, the term efficiency is connected with the resources of the computer (time and space) that are occupied by running an algorithm. Generally, the most efficient algorithm is the one that finds the solution to the problem in the fastest way.

In practice, the time complexity of an algorithm is not measured by the effective time necessary for solving the problem on a concrete computer because this

measurement suffers from a lack of objectiveness. The same algorithm could be run on different hardware configurations or even on different operating systems. Therefore, the algorithm's complexity is measured in an informal way that determines the complexity with regard to the amount of input data (instance size), necessary for the problem description. In the case of TSP, the complexity of the problem is determined by the number of cities that the traveling salesman must visit.

The time complexity of an algorithm determines the way in which the increase in the instance size influences the time complexity (Garey and Johnson, 1979). This relation can be expressed with the so-called asymptotic time complexity function $O(\hat{f}(n))$ that determines the upper bound of time complexity for problem P. For example, the function $O(n^2)$ denotes that the increase in the instance size n will cause an increase in the time complexity to almost n^2.

When do the problems become hard (Garey and Johnson, 1979)? The algorithmic theory divides problems, with regard to the asymptotic time complexity function, into two classes: P-hard and NP-hard. To the first class belong those problems that have polynomial time complexity $O(n^k)$ and are treated as "easy." In contrast, problems of class NP-hard demonstrate the exponential time complexity $O(2^n)$ and are, therefore, treated as "hard." That is, the exponential time complexity may cause that some increase in the input data can increase solution time of the problem exponentially. In the worst case, we could be waiting for the solution over an infinite period of time.

4.3 Memetic Self-Adaptive Firefly Algorithm

Fireflies can generate flashing light that can be admired on clear summer nights. The flashing light is the result of a process of bioluminescence that comprises a complicated set of chemical reactions. The purpose of firefly flashing may be still debating; however, it can be twofold: to attract possible mating partners (communication) and to warn off potential predators.

The light intensity I_L of a flashing firefly decreases as the distance from source r increases in terms of $I_L \propto 1/r^2$. Additionally, air absorption causes the light to become weaker and weaker as the distance from the source increases. The flashing light represents the inspiration for the development of the FFA by Yang (2008). Here, light intensity is proportional to the fitness function of the problem being optimized (i.e., $I_L(\mathbf{s}) \propto f(\mathbf{s})$, where $\mathbf{s} = S(\mathbf{x})$ represents a candidate solution).

In order to formulate the FFA, some flashing characteristics of the fireflies are idealized, as follows:

- All fireflies are unisex.
- Their attractiveness is proportional to their light intensity.
- The light intensity of a firefly is affected or determined by the landscape of the fitness function.

Note that light intensity I_L and attractiveness β are in some way synonymous. While the intensity I_L is referred to as an absolute measure of emitted light by the firefly, the attractiveness β is a relative measure of the light that should be seen in

the eyes of the beholders and judged by other fireflies (Yang, 2008). The light intensity I_L varied by distance r is expressed by the following equation

$$I_L(r) = I_{L_0}\, e^{-\gamma r^2} \tag{4.1}$$

where I_{L_0} denotes the intensity light at the source and γ is a fixed light absorption coefficient. Similarly, the attractiveness β also depends on the distance r that is calculated according to the following generalized equation

$$\beta(r) = \beta_0\, e^{-\gamma r^k}, \quad \text{for } k \geq 1 \tag{4.2}$$

The distance between two fireflies i and j is represented as the Euclidian distance:

$$r_{ij} = ||\mathbf{x}_i - \mathbf{x}_j|| = \sqrt{\sum_{k=1}^{n}(x_{ik} - x_{jk})} \tag{4.3}$$

where x_{ik} is the kth element of the ith firefly's position within the search space. Each firefly i moves to another more attractive firefly j as follows:

$$\mathbf{x}_i = \mathbf{x}_i + \beta_0\, e^{-\gamma r_{ij}^2}(\mathbf{x}_j - \mathbf{x}_i) + \alpha\, N_i(0,1) \tag{4.4}$$

Equation (4.4) consists of three terms. The first term determines the current position of ith firefly. The second term refers to the attractiveness, while the third term is connected with the randomized movement of ith firefly within the search space. This term consists of the randomization parameter α, and random numbers $N_i(0,1)$ drawn from a Gaussian distribution. The scheme of FFA is sketched in Algorithm 4.1.

The FFA (Algorithm 4.1) runs on the population of fireflies $P^{(t)}$ that are represented as real-valued vectors $\mathbf{x}_i^{(t)} = (x_{i0}^{(t)}, \ldots, x_{in}^{(t)})$, where $i = 1, \ldots, NP$ and NP denotes the number of fireflies within a population $P^{(t)}$ at generation t. Note that each firefly $\mathbf{x}_i^{(t)}$ is of dimension n. The population of fireflies is initialized randomly (function *InitFFA*) according to the following equation:

$$x_{ij}^{(0)} = (\text{ub} - \text{lb}) \cdot \text{rand}(0,1) + \text{lb} \tag{4.5}$$

where ub and lb denote the upper and lower bounds, respectively. The main loop of the firefly search process that is controlled using the maximum number of generations MAX_GEN consists of the following functions:

- *AlphaNew*: Calculating new values for randomization parameter α. This parameter is modified according to the equation as follows:

$$\Delta = 1 - (10^{-4}/0.9)^{1/\text{MAX\_GEN}}, \quad \alpha^{(t+1)} = (1 - \Delta) \cdot \alpha^{(t)} \tag{4.6}$$

  where Δ determines the step size when changing the parameter $\alpha^{(t+1)}$. Note that this parameter decreases with the increasing of generation counter t.

```
01: algorithm Firefly
02: t = 0; x* = ∅; γ = 1.0;                    // initialize: gen.counter, best solution, attractiveness
03: P(t) = InitializeFFA();                    // initialize a population x_i(0) ∈ P(0)
04: while (t ≤ MAX_GEN) do                     // termination condition
05:     α(t) = AlphaNew();                     // determine a new value of α
06:     EvaluateFFA(P(t), f(x));               // evaluate x_i(t) according to f(x_i)
07:     OrderFFA(P(t), f(x));                  // sort P(t) according to f(x_i)
08:     x* = FindTheBestFFA(P(t), f(x));       // determine the best solution x*
09:     P(t+1) = MoveFFA(P(t));                // vary attractiveness according Equation (4)
10:     t = t+1;                               // next generation
11: end while
12: return x*, f(x);                           // postprocess results
```

Algorithm 4.1 Pseudo code of the FFA.

- *EvaluateFFA*: Evaluating the new solution $\mathbf{x}_i^{(t)}$ according to a fitness function $f(\mathbf{s}_i^{(t)})$, where $\mathbf{s}_i^{(t)} = S(\mathbf{x}_i^{(t)})$.
- *OrderFFA*: Ordering solutions $\mathbf{x}_i^{(t)}$ for $i = 1, \ldots, NP$ with respect to the fitness function $f(\mathbf{s}_i^{(t)})$ ascending, where $\mathbf{s}_i^{(t)} = S(\mathbf{x}_i^{(t)})$.
- *FindTheBestFFA*: Determining the best solution within the population $P^{(t)}$. Normally, the best solution becomes $\mathbf{x}^* = \mathbf{x}_0^{(t)}$.
- *MoveFFA*: Moving the fireflies toward the search space according to the attractiveness of their neighbor's solution (Eq. (4.4)).

In order to improve the original FFA, especially in the sense of exploration and exploitation, the following features are incorporated into our new MSA-FFA:

- self-adaptation of control parameters,
- new population scheme,
- more directly controlling the balance between exploration and exploitation during the search process,
- hybridization using local search heuristics.

These features will be discussed in more detail in the remainder of this chapter.

4.3.1 Self-Adaptation of Control Parameters

Population-based algorithms often use the information explored to a certain generation within a population (Bäck, 1998). However, their efficiency depends on the

characteristics of the population diversity, i.e. accumulated information about the problem that is written into the genotypes of individuals. The greater the diversity of the population the greater the search power of the population-based algorithm. In summary, how to reach the information accumulated into a population depends primarily on the appropriate setting of the control parameters.

Unfortunately, setting parameters that are appropriate at the beginning of optimization can become inappropriate for later generations. Therefore, the idea of adapting control parameters during optimization arose (Holland, 1992). This idea has overgrown into the self-adaptation of control parameters, where these are encoded into genotypes of individuals and undergo operations of the variation operators (Meyer-Nieberg and Beyer, 2007).

It is worth pointing out that FFA involves three control parameters: the randomization parameter α, the attractiveness β, and the light absorption coefficient γ. All the mentioned parameters are encoded into real-valued vectors in the following form:

$$\mathbf{x}_i^{(t)} = \langle x_{i0}^{(t)}, \ldots, x_{in}^{(t)}; \alpha^{(t)}, \sigma_0^{(t)}; \beta^{(t)}, \sigma_1^{(t)}; \gamma^{(t)}, \sigma_2^{(t)} \rangle, \quad \text{for } i = 1, \ldots, NP \tag{4.7}$$

where the first part of vector $\mathbf{x}_i^{(t)} = (x_{i0}^{(t)}, \ldots, x_{in}^{(t)})$ represents a position of the ith firefly similar to the original FFA, parameters $\alpha^{(t)}$, $\beta^{(t)}$, and $\gamma^{(t)}$ are the current values of the control parameters, while $\sigma_0^{(t)}$, $\sigma_1^{(t)}$, and $\sigma_2^{(t)}$ refer to their standard deviations (also mutation strengths). Interestingly, the first part is changed according to Eq. (4.4), while the self-adaptive parameters undergo an operation of uncorrelated mutation with three-step sizes (Eiben and Smit, 2003). This mutation is described by the following equations:

$$\sigma_i^{(t+1)} = \sigma_i^{(t)} \cdot \mathrm{e}^{(\tau' \cdot N(0,1) + \tau N_i(0,1))}, \quad \text{for } i = 1, \ldots, 3 \tag{4.8}$$

$$\alpha^{(t+1)} = \alpha^{(t)} + \sigma_0^{(t)} \cdot N(0,1)$$

$$\beta^{(t+1)} = \beta^{(t)} + \sigma_1^{(t)} \cdot N(0,1)$$

$$\gamma^{(t+1)} = \gamma^{(t)} + \sigma_2^{(t)} \cdot N(0,1)$$

where $\tau' \propto 1/\sqrt{2n}$ and $\tau \propto 1/\sqrt{2 \cdot \sqrt{n}}$ denote the so-called learning rate. Here, the rule which prevents that the mutation strengths $\sigma_i^{(t)}$ do not fall under a certain minimum value ε_0 is applied:

$$\sigma_i^{(t)} < \varepsilon_0 \Rightarrow \sigma_i^{(t)} < \varepsilon_0, \quad \text{for } i = 1, \ldots, 3 \tag{4.9}$$

4.3.2 Population Model

The original FFA uses a generational population model, where the entire population is replaced in each generation. Specifically, *NP* parents produce *NP* offspring that

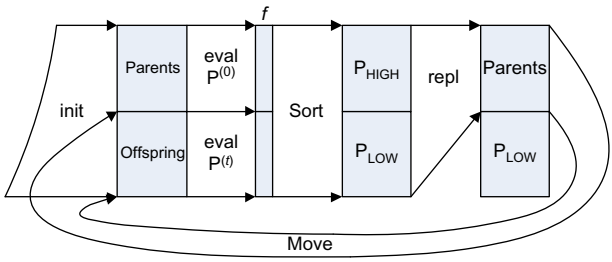

Figure 4.2 Population model.

become parents in the next generation. Here, no selection pressure influences the survival of individuals. In fact, each parent is alive for one generation only. Furthermore, the best solution is not preserved by FFA, i.e., elitism is not used in the original FFA.

Our new population model is presented in Figure 4.2. It can be seen from this figure that the initialized population forms an initial population consisting of two *NP* individuals, where the first *NP* individuals represent a subpopulation of parents, while the next *NP* individuals form a subpopulation of the offspring. Firstly, individuals from the whole population are evaluated (EvalFFA). Second, both the parent and the offspring are placed in ascending order according to their fitness values (OrderFFA). Therefore, the whole population representing a mating pool is divided into high and low subpopulations. The former consists of individuals with the higher fitness, while the latter have the lower fitness values irrespective of whether an individual is either a parent or an offspring. Third, individuals from both subpopulations are replaced by the high subpopulation (ReplaceFFA). Note that the high subpopulation consists of overfit individuals and, therefore, in some way represents an exploitation subpopulation. On the other hand, the low, more explorative subpopulation takes care of diversity. During this phase, the balance between exploration and exploitation can be realized. Fourth, individuals from the high subpopulation represent the parents for the next generation that are reproduced in the sense of Eq. (4.4) (MoveFFA). Finally, the offspring that occupy the low subpopulation enter into the next generation of the search process together with their parents.

Note that this population model is elitist because it ensures that the best individuals are always preserved.

4.3.3 Balancing Between Exploration and Exploitation

The original FFA automatically subdivides the population (swarm) in a problem's search space into subgroups (swarms), where each subgroup explores its own region within this space. As a result, multiple optima can be found simultaneously. Unfortunately, there is no warranty that the global optimum is found by at least one subgroup, unless in the case when the number of fireflies are much higher than

the number of modes. Stagnation of FFA can often be detected during the search process if the randomness is reduced too quickly. Furthermore, the population can lose diversity resulting in premature convergence. These phenomena are not only connected with a fitness landscape, as determined by the problem, but also with the nature of the swarm intelligence framework for almost all swarm-based algorithms. Namely, the search process behaves differently if it runs within either evolutionary or swarm intelligence frameworks. That is, a certain conclusion that is valid for the former does not always hold for the latter and vice versa (Neri, 2012).

The proposed MSA-FFA employs a fitness diversity adaptation for preventing the loss of population diversity and balancing the exploration and exploitation. The fitness diversity adaptation methods measure the fitness diversity in order to estimate the population diversity. On the other hand, these measures can also serve for balancing between exploration and exploitation in the search process. Interestingly, the fitness diversity refers to the phenotype level, where these measures can be more efficiently calculated than similar measures on the genotype level.

The fitness diversity metric used in MSA-FFA is defined as follows (Neri et al., 2007):

$$\Psi = 1 - \left| \frac{f_{\text{avg}} - f_{\text{min}}}{f_{\text{max}} - f_{\text{min}}} \right| \tag{4.10}$$

where f_{min}, f_{avg}, and f_{max} are, respectively, the minimum, average, and maximum fitness values within the population. This measure determines where the average fitness is positioned between the minimum and maximum fitness variations in the current population. This metric can occupy any value in the interval $\Psi \in [0, 1]$. That is, when the value of the metric Ψ is close to zero the population diversity is low, while the population diversity is high when this value is close to one. This metric is very sensitive to small variations in the fitness landscapes with plateaus and low-gradient areas (Neri, 2012).

In order to measure population diversity on a genotype level, the concept of moment of inertia is applied to MSA-FFA (Morrison, 2004). The moment of inertia is a measure of how the mass of the individuals is distributed from the center of gravity. In our case, the center of gravity $\overline{\mathbf{x}} = (\overline{\mathbf{x}}_1, \ldots, \overline{\mathbf{x}}_n)$ (also the centroid) is expressed as follows:

$$\overline{x}_i = \frac{1}{NP} \cdot \sum_{j=1}^{NP} x_{ij} \tag{4.12}$$

where $x_{ij} \in \mathbb{R}$, NP is the population size and $\overline{x}_i$ the ith element of the centroid. The moment of the inertia diversity measure is defined as follows:

$$I_{\text{C}} = \sum_{i=1}^{n} \sum_{j=1}^{NP} (x_{ij} - \overline{x}_i)^2 \tag{4.13}$$

where n is a dimensionality of the problem. As a measure of similarity between two individuals at the genotype level, the correlation between individual $\mathbf{x}_i$ and centroid $\overline{\mathbf{x}}$ is introduced as follows:

$$K_i = \frac{1}{n} \cdot \sum_{j=1}^{n} x_{ij} \cdot \overline{x}_j - \overline{\mathbf{x}}_i \cdot \overline{\mathbf{x}} \tag{4.14}$$

where $\overline{\mathbf{x}}_i = \frac{1}{n} \cdot \sum_{j=1}^{n} x_{ij}$, and $\overline{x}_j$ denotes the jth element of the centroid.

At each generation, the fitness diversity Ψ is calculated and then MSA-FFA uses this information to coordinate exploration and exploitation within the search. Let us suppose that the new population model of MSA-FFA divides the population $\mathrm{P}^{(t)}$ into two subpopulations: $\mathrm{P}^{(t)}_{\mathrm{HIGH}}$ and $\mathrm{P}^{(t)}_{\mathrm{LOW}}$. Individuals in population $\mathrm{P}^{(t)}_{\mathrm{HIGH}}$ are overfit and tend to exploit the explored information held into population, as soon as possible. These individuals are ordered according to their fitness ascendency. Usually, this exploitation causes a loss of population diversity that can lead to premature convergence. The subpopulation $\mathrm{P}^{(t)}_{\mathrm{LOW}}$ consists of underfit individuals but maintains higher population diversity. Individuals within this subpopulation are ordered according to their descending covariance (Eq. (4.14)). The proper selection of individuals from both subpopulations may slow down the fast exploitation process by FFA and direct the search process toward underfit individuals in order to explore new, probably more promising, regions of the search space. Thus, it is expected that population diversity should not decrease too fast so as to maintain a good balance.

Balancing between exploration and exploitation into the MSA-FFA is performed according to the following equation:

$$\mathrm{P}_i^{(t+1)} = \begin{cases} (r < (0.5 - \Psi)) \text{ and } (i \neq 0) & \Rightarrow \quad \mathrm{P}^{(t)}_{\mathrm{LOW}} \\ \text{otherwise} & \Rightarrow \quad \mathrm{P}^{(t)}_{\mathrm{HIGH}} \end{cases}, \text{ for } i = 1 \ldots NP \tag{4.11}$$

where r denotes the randomly generated number from interval $[0, 1]$. Note that this equation is implemented into the procedure ReplaceFFA in Algorithm 4.2 and acts as follows: When the metric Ψ is calculated, a selection of those individuals is started that can progress to the next generation. For each individual, a random number $r \in [0, 1]$ is generated. If the generated number r is lower than the factor $(0.5 - \Psi)$, the first not-used individual from the population $\mathrm{P}^{(t)}_{\mathrm{LOW}}$ is taken, otherwise on the same position laid individual from the population $\mathrm{P}^{(t)}_{\mathrm{HIGH}}$ is preserved. In the former case, the population diversity is being decreased. Therefore, the underfit individuals with the highest covariance are selected for the next generation thus caused an increase in the population diversity. In the latter case, the overfit individuals can progress to the next generation. Here, the value 0.5 represents a threshold that balances the exploration and exploitation components within firefly search. Note that in the worst case when the population diversity is lost, half individuals from both subpopulations contribute as candidates for the new population.

01: **algorithm** MemeticFirefly

02: $t = 0$; $fe = 0$; $\mathbf{x}^* = \varnothing$; // initialize: gen.counter, fun.eval.counter, best solution

03: $P^{(t)}$ = *InitializeFFA*(); // initialize a population $\left\langle \mathbf{x}_i^{(0)}; \alpha^{(0)}, \sigma_0^{(0)}; \beta^{(0)}, \sigma_1^{(0)}; \gamma^{(0)}, \sigma_2^{(0)} \right\rangle \in P^{(0)}$

04: **while** ($fe \leq$ MAX_FE) **do** // termination condition

05: *SelfAdaptFFA*(); // self-adapting the control parameters

06: fe += *EvalAndImproveFFA*($P^{(t)}$, $f(\mathbf{x})$); // evaluate and improve $\mathbf{x}_i^{(t)}$ according to $f(\mathbf{x}_i)$

07: *OrderFFA*($P^{(t)}$, $f(\mathbf{x})$); // sort $P^{(t)}$ according to $f(\mathbf{x}_i)$

08: *ReplaceFFA*($P_{HIGH}^{(t)}$, $P_{LOW}^{(t)}$); // replace $P_{HIGH}^{(t)}$ with individuals of $P^{(t)}$

09: $\mathbf{x}^*$ = *FindTheBestFFA*($P_{HIGH}^{(t)}$, $f(\mathbf{x})$); // determine the best solution $\mathbf{x}^*$

10: $P^{(t+1)}$ = *MoveFFA*($P_{HIGH}^{(t)}$); // vary attractiveness according Equation (4)

11: $t = t+1$; // next generation

12: **end while**

13: **return** $\mathbf{x}^*$, $f(\mathbf{x})$; // postprocess results

Algorithm 4.2 Pseudo code of the MSA-FFA.

4.3.4 The Local Search

Local search is a kind of improvement heuristics when used properly and effectively. The main characteristic of these heuristics is that the solution is not created anew but by improving the current solution. The local search is defined as an iterative process of exploiting the search region within neighborhood of the current solution. If a better new solution is found, then the current one is replaced by it (Aarts and Lenstra, 1997; Bäck, 1998). The neighborhood of current solution $\mathbf{x}$ is defined as a set of solutions that can be reached by an elementary operator, i.e., $N : S \rightarrow 2^S$. Each solution in their neighborhood N is accessed from the current solution in k-moves. In line with this, these solutions represent the k-neighborhood as well.

The crucial point of the local search algorithm represents a transformation of the current solution through an elementary operator (Hoos and Stützle, 2005). In the case where the number of moves is increased, the operator creates solutions almost randomly. On the other hand, when the number of moves is small, the operator can even return the same solution quite often. Thus, no improvement of the current solution is detected. In a practice, the proper transformation is found somewhere in the middle.

Another crucial step is performed according to the size of the neighborhood. When the neighborhood is small, the solution can be found quickly, but it is possible that the local search algorithm may get stuck in the local minimum. In contrast, when the neighborhood is too large, searching for solutions can take too long.

4.3.5 Scheme of the MSA-FFA

The scheme of the MSA-FFA is presented in Algorithm 4.2.

The MSA-FFA runs on a population of fireflies $P^{(t)}$ that, beside the real-valued problem, variables also encode their control parameters as follows. $\mathbf{x}_i^{(t)} = \langle x_{i0}^{(t)}, \ldots, x_{in}^{(t)}; \alpha^{(t)}, \sigma_0^{(t)}; \beta^{(t)}, \sigma_1^{(t)}; \gamma^{(t)}, \sigma_2^{(t)} \rangle$ for $i = 1, \ldots, 2NP$, where NP denotes the number of fireflies in two subpopulations, i.e., subpopulation of parents and subpopulation of offspring. The problem variables $(x_{i0}^{(t)}, \ldots, x_{in}^{(t)})$ are initialized randomly according to Eq. (4.5), while the control parameters $\langle \alpha^{(t)}, \sigma_0^{(t)}; \beta^{(t)}, \sigma_1^{(t)}; \gamma^{(t)}, \sigma_2^{(t)} \rangle$ are set initially to prescribed values.

Note that the setting values of mutation strengths $\sigma_i^{(0)}$ for $i = 1,\ldots,3$, play especially an important role by a self-adaptive search process. These values determine a region in which the search process can progress. In fact, this search process can progress within the limited interval of these values, also called a progress window. Unfortunately, this region is unknown in advance and usually determined through extensive experimental work, in practice.

The main loop of MSA-FFA incorporates the following functions:

- *SelfAdaptFFA*: Self-adaptation of control parameters according to Eq. (4.8).
- *EvalAndImproveFFA*: Evaluating the new solution $\mathbf{x}_i^{(t)}$ for $i = NP, \ldots, 2NP$ with respect to a fitness function $\boldsymbol{f}$. Note that in the first generation ($t = 0$), the whole population is evaluated, i.e., for $i = 1, \ldots, 2NP$. Each solution is improved in the sense of local search heuristics. However, any implementation of these heuristics depends on the problem to be solved. The number of fitness evaluations depends on the local search heuristic and is unknown in advance. Therefore, this number is obtained as a return value from the procedure.
- *OrderFFA*: Sorting solutions $\mathbf{x}_i^{(t)}$ for $i = 1, \ldots, NP$ ascending with regard to the fitness function $\boldsymbol{f}$ and for $i = NP, \ldots, 2NP$ descending with regard to the correlations K_i, thus dividing the population $P^{(t)}$ in $P_{\text{HIGH}}^{(t)}$ and $P_{\text{LOW}}^{(t)}$
- *ReplaceFFA*: Selecting parents from both subpopulations $P_{\text{HIGH}}^{(t)}$ and $P_{\text{LOW}}^{(t)}$ and thus balancing between exploration and exploitation. In the sense of elitism, the best solution is preserved in the parents' population.
- *FindTheBestFFA*: Determining the best solution within the population $P^{(t)}$.
- *MoveFFA*: Moving the fireflies toward the search space according to the attractiveness of their neighbor's solution (Eq. (4.4)).

The proposed algorithm terminates when the number of fitness function evaluations exceeds the maximum number of fitness function evaluations MAX_FE.

4.4 Case Study: Graph 3-Coloring

In order to show how the proposed hybridizations of the original FFA influence the results of MSA-FFA, we have carried out a case study by solving the well-known 3-GCP on large-scale graphs (graphs with 1000 vertices). This problem is a well-known, tough combinatorial optimization problem. Note that this kind of optimization problem was seldom solved by swarm intelligence−based algorithms, and this work is among the first attempts.

Graph coloring can informally be defined as follows: How to color an undirected graph $G = (V, E)$, where V represents a set of vertices and E a set of edges (unordered pairs of vertices) with almost k colors so that neither of the two vertices connected with an edge is colored with the same color. If such coloring exists, it is also named as proper k-coloring (Bondy and Murty, 2008). The problem of finding the proper k-coloring is denoted as k-GCP. The minimum number of colors k for which proper coloring exists is also known as the chromatic number χ of graph G. 3-GCP is a special kind of common k-coloring where the number of colors is limited to 3 ($k = 3$). The complexity of the k-GCP is determined as follows: The decision form of this problem is NP-complete, while the construction form is NP-hard (Garey and Johnson, 1979).

To solve this problem exactly, i.e., by enumerating all possible solutions, is only limited to the instances of graphs with less than 100 vertices because of its complexity. Instead, several heuristic methods that approximately solve the problem have emerged in the past. The simplest way of coloring the vertices of graph G is in a greedy fashion. Thus, vertices are ordered in a permutation and colored sequentially one after another. However, the quality of this so-called sequential coloring depends on the permutations of vertices. In order to find a more promising sequence of vertices, many methods for ordering the permutation of vertices have been incorporated into sequential graph coloring algorithms. For example, vertices are ordered randomly by the naive method (Kubale, 2004). Much better ordering can be applied by the DSatur traditional heuristic (Brelaz, 1979), where the vertices are dynamically ordered according to their saturation degrees ρ_v. The saturation degree is defined as the number of distinctly colored vertices adjacent to vertex v (Bondy and Murty, 2008).

Today, some of the most popular algorithms for solving k-GCP are metaheuristics based on local search (Blum and Roli, 2003; Blum et al., 2011). One of the first such metaheuristics was developed by Hertz and de Werra (1987), known under the name Tabucol. At the same time, this was the first application of the Tabu search (Glover, 1986) to graph coloring. Tabucol acts as follows: at first, it generates an initial random k-coloring, which typically contains a large number of conflicting edges. Then, the heuristic iteratively looks for a single vertex that the most decreases the number of conflicting edges when it is recolored with another color, i.e., moved to another color class. A Tabu list prevents the moves from cycling. Proper k-coloring may be obtained after a definite number of iterations. Later, Tabucol was improved by more sophisticated graph coloring algorithms (Dorne and Hao, 1998; Galinier and Hao, 1999).

Other local search heuristics include simulated annealing (Chams et al., 1987; Johnson et al., 1991), iterative local search (Chiarandini and Stützle, 2002; Chiarandini et al., 2007), reactive partial Tabu search (Blöchliger and Zufferey, 2003, 2008; Malaguti et al., 2008), variable neighborhood search (Avanthay et al., 2003), adaptive memory (Galinier et al., 2008), variable search space (Hertz et al., 2008), and population-based methods (Dorne and Hao, 1998; Fleurent and Ferland, 1996; Galinier and Hao, 1999; Lü and Hao, 2010). One of the best population-based algorithms for k-GCP, the hybrid EA (HEA), developed by Galinier and Hao (1999)

combines the local search with the partition-based crossover operator. Here, the Tabucol metaheuristic is used as a local search operator. Recently, a distributed graph coloring algorithm that was inspired by the calling behavior of Japanese tree frogs was proposed by Hernández and Blum (2012). For a comprehensive survey of the main methods, see, for example, Galinier and Hertz (2006) and Malaguti and Toth (2009).

This case study was a continuation of work presented by Fister et al. (2012a,b) that introduced the memetic FFA (MFFA) for 3-GCP. This paper was inspired by the hybrid self-adaptive EA of Fister et al. (2013), the hybrid self-adaptive differential evolution of Fister and Brest (2011), and the hybrid ABC of Fister et al. (2012a,b). However, the common characteristic of all these algorithms was solving of the same problem, i.e., the 3-GCP.

MFFA (Fister et al., 2012a,b) exposed excellent results when coloring the medium-scale graphs. This algorithm operates on real-valued vectors whose elements represent weights that determine how hard the vertex is to color. The higher the weight is, the faster it needs to be colored. An initial permutation of vertices is obtained when the vertices are ordered according to the weights. This permutation serves as an input to DSatur traditional heuristic that constructs corresponding 3-coloring. A similar approach was used by Eiben et al. (1998) that developed the EA with a stepwise adaptation of the weights method (EA-SAW), in order to solve the 3-GCP. Instead of DSatur, the greedy algorithm was used for the construction of 3-coloring by the authors of EA-SAW. Two additional features have been applied to MFFA as follows: heuristic swap local search and elitism.

The preliminary results of MFFA for 3-GCP on large-scale graphs were not promising. Over several runs the search process of FFA was detected either as stagnation or as premature convergence. Therefore, the MSA-FFA was proposed, that tries to overcome these drawbacks of MFFA, using the following features:

- self-adaptation of control parameters,
- new population model,
- balancing between exploration and exploitation.

A formal definition of graph 3-coloring is firstly provided in the remainder of this section. Then, the characteristics of MSA-FFA for 3-GCP are described. Next, the experiments and results are illustrated. Finally, the results of the experimental work are summarized and discussed.

4.4.1 Graph 3-Coloring

Graph 3-coloring is formally defined as follows: An undirected graph $G = (V, E)$ is given, where V is a set of vertices $v \in V$ for $i = 1, \ldots, n$ and E denotes a set of edges that associate each edge $e \in E$ for $j = 1, \ldots, m$ to the unordered pair $e = \{v_i, v_j\}$. Then, the vertex 3-coloring is defined as a mapping $c : V \rightarrow S$, where $S = \{1, 2, 3\}$ is a set of three colors and c is a function that assigns one of the three colors to each vertex of G. A coloring $\mathbf{s}$ is proper if each of the two vertices connected with an edge is colored with a different color.

Interestingly, 3-GCP belongs to a class of CSPs. Each CSP is represented as a pair $\langle S, \phi \rangle$, where S denotes the search space of feasible solutions $\mathbf{s} \in S$ and ϕ is the feasibility condition on S that divides the search space into feasible and unfeasible regions. To each $e \in E$, the constraint b_e is assigned with $b_e(\langle s_1, \ldots, s_n \rangle) = \text{true}$ if and only if $e = \{v_i, v_j\}$ and $s_i \neq s_j$. Suppose that $B^i = \{b_e | e = \{v_i, v_j\} \wedge j = 1, \ldots, m\}$ defines the set of constraints belonging to variable v_i. Then, the feasibility condition ϕ is expressed as a conjunction of all the constraints $\phi(\mathbf{s}) = \wedge_{v \in V} B^v(\mathbf{s})$.

As in evolutionary computation, constraints can be handled indirectly in the sense of a penalty function that punishes the unfeasible solutions. The further the unfeasible solution is from the feasible region, the higher the penalty function. The penalty function is expressed as

$$f(\mathbf{s}) = \min \sum_{i=0}^{n} \psi(\mathbf{s}, B^i) \tag{4.14}$$

where the function $\psi(\mathbf{s}, B^i)$ is defined as

$$\psi(\mathbf{s}, B^i) = \begin{cases} 1 & \text{if } \mathbf{s} \text{ violates at least one } b_e \in B^i \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

Note that Eq. (4.1) also represents the objective function. On the other hand, the same equation can be used as a feasibility condition in the sense that $\phi(\mathbf{s}) = \text{true}$ if and only if $f(\mathbf{s}) = 0$. The proper graph 3-coloring is found if this condition is satisfied.

4.4.2 MSA-FFA for Graph 3-Coloring

The MSA-FFA consists of the following components and features:

- representation of individuals,
- self-adaptation of control parameters,
- evaluation of fitness function,
- population model,
- balancing between exploration and exploitation,
- moves of individuals,
- initialization procedure,
- termination condition.

Each individual in MSA-FFA is composed of problem variables and control parameters according to Eq. (4.7). Problem variables that determine points in the fitness landscape represent those weights from which an initial permutation of vertices is built by the DSatur traditional heuristic. Control parameters are self-adapted according to Eq. (4.8). In this algorithm, the new population model is implemented as discussed in Section 4.3.2. Additionally, the exploration and exploitation are balanced as proposed in Section 4.3.3, while the individuals are moved through the search

space according to Eq. (4.4). The population is initialized according to Eq. (4.5). The search process is terminated, when the proper 3-coloring is found or the maximum number of fitness function evaluations exceeds MAX_FE.

Evaluation of the fitness function is crucial for the results of optimization. This function is problem dependent. On the other hand, an FFA belongs to a kind of general problem solvers, where the good results should be obtained independently of the problem to be solved. Although this algorithm can be applied to several real-world optimization problems, its performance is subject to the No Free Lunch (NFL) theorem (Wolpert and Macready, 1997). According to this theorem, any two algorithms are equivalent in the sense of average performance, when their performance is compared across all possible problems. Fortunately, the NFL theorem can be circumvented for a given problem by hybridization that incorporates the problem-specific knowledge into FFAs.

In the case of MSA-FFA, the problem-specific knowledge can be conducted by the evaluation of the fitness function. In other words, the fitness function is hybridized with the domain-specific knowledge. Two kinds of hybridization are implemented into this, as follows:

1. hybrid genotype−phenotype mapping,
2. heuristic swap local search.

These hybridizations are described in detail in the remainder of this section.

4.4.2.1 *Hybrid Genotype−Phenotype Mapping*

Many problems are hard to represent within their original problem context. For instance, genetic algorithms (Goldberg, 1989) operate on the population of binary vectors but can also be successfully applied to continuous optimization problems. In that case, a transformation from the binary represented variables to the real-valued solution must be performed. A solution in its problem context is referred to as a phenotype, while the same solution in its encoded form is a genotype. Transformation from a solution in encoded form to a solution in problem context is known as genotype−phenotype mapping.

In MSA-FFA, the genotype is represented as real-valued vector of weights that determine an initial permutation of vertices, while the phenotype determines the graph 3-coloring obtained by DSatur traditional heuristic. The quality of solution is calculated according to Eq. (4.14). The hybrid genotype−phenotype mapping in MSA-FFA transforms the vector of weights into 3-coloring. This mapping consists of two phases:

1. Ordering the vertices $\mathbf{v}_i^{(t)} = (v_{i0}^{(t)}, \ldots, v_{in}^{(t)})$ with regard to weights $\mathbf{x}_i^{(t)} = (x_{i0}^{(t)}, \ldots, x_{in}^{(t)})$ descending and thus determining a permutation of vertices $\Pi(\mathbf{v}_i^{(t)})$.
2. From the permutation of vertices $\Pi(\mathbf{v}_i^{(t)})$ finding the graph 3-coloring $\mathbf{s}_i^{(t)} = (s_{i0}^{(t)}, \ldots, s_{in}^{(t)})$ due to traditional heuristic DSatur and evaluating it according to Eq. (4.14).

Note that the genotype space is much bigger than the phenotype space. The former is determined by the size of the permutation space that can be obtained with

n vertices, i.e., $n!$, while the latter is estimated by the size of the combinatorial space that can be obtained using 3-colors, i.e., 3^n. Unfortunately, inspecting the genotype search space is much easier to implement in an FFA than inspecting the phenotype space because of the many heuristics that are available for exploring the permutation search space.

4.4.2.2 The Heuristic Swap Local Search

When a solution is evaluated by the MFFA, the heuristic swap local search tries to improve it. This heuristic is run until the improvements are detected. The operation of this is illustrated in Figure 4.3, which deals with a solution on G with 9 vertices. This solution is composed of a permutation of vertices **v**, 3-coloring **s**, weights **y**, and saturation degrees **ρ**. The heuristic swap local search takes the first uncolored vertex in a solution and orders the predecessors according to the descending saturation degree. The uncolored vertex is swapped with the vertex that has the highest saturation degree. In the case of a tie, the operator randomly selects a vertex among the vertices with higher saturation degrees (2-opt neighborhood).

In Figure 4.3, an element of the solution corresponding to the first uncolored vertex 5 is in dark gray and the vertices 0 and 3 with the highest saturation degree are in light gray. From vertices 0 and 3, heuristical swap randomly selects vertex 0 and swaps it with vertex 5 (the right-hand side of Figure 4.3).

4.4.3 Experiments and Results

The goal of the experimental work is to show that MSA-FFA can be successfully applied to 3-GCP on large-scale graphs. As this work continues the experiments as represented in the paper Fister et al. (2012a,b), the same test algorithms were also used in this comparative study, i.e., Chiarandini and Stützle (2012) implementations of HEA and Tabucol and van Hemert (2012) implementation of EA-SAW. In order to help the developers of a new graph coloring algorithms, the authors put these implementations on the Internet to make a comparison with the newly developed algorithms as fairly as possible. Additionally, the basic FFA was also included into comparison in order to obtain suitable conclusions.

The characteristics of MSA-FFA in the experiments were as follows: The population size was fixed at 20. MAX-FE was set at 300,000 by all algorithms to make

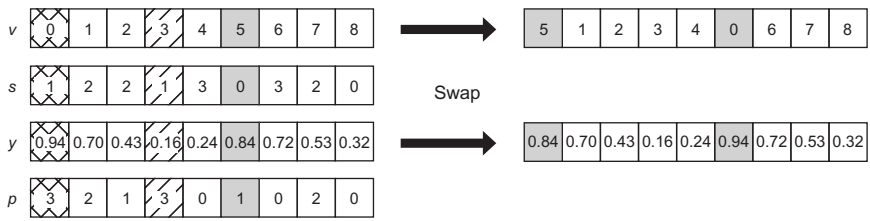

Figure 4.3 Heuristic swap local search.

the fair comparison with the paper of Eiben et al. (1998). Each instance of the graph was executed 25 times. The initial values of the self-adaptive control parameters were set as follows: $\alpha^{(0)} = 0.1$, $\beta^{(0)} = 0.1$, and $\gamma^{(0)} = 0.8$, while the lower and upper bounds of these parameters were self-adapted over the following intervals: $\alpha^{(t)} \in [0.001, 0.2]$, $\beta^{(t)} \in [0.001, 0.2]$, and $\gamma^{(t)} \in [0.1, 1.0]$. Mutation strengths were initially set as $\sigma_0^{(t)} = \sigma_1^{(t)} = \sigma_2^{(t)} = 0.03$, while the minimum value of mutation strengths was limited by $\varepsilon_0 = 0.0001$. The same values of control parameters were also used by the basic FFA. However, these parameters were fixed during the experiments.

The algorithms were compared according to the measures success rate (SR) and average estimations to solution (AES). The former determines how successfully the particular algorithm is by solving the given instance of the graph. It is expressed as a ratio between the number of successfully runs and the number of all runs. The latter determines the efficiency of the particular algorithm and counts the average number of fitness function evaluations needed to find the solution.

Three experiments were conducted during this work in order to show how parameters edge probability and the fitness diversity metric Ψ influence the results of optimization. Additionally, an analysis of the inertia diversity metric is presented during the arbitrary search process.

4.4.3.1 Test Suite

The test suite during the experiments consists of graphs generated using the Culberson random graph generator (Culberson, 2012). The graphs generated by this generator are distinguished according to type, number of vertices n, edge probability p, and the seeds of random number generator q. Three types of graphs were employed during the experiments: uniform (random graphs without variability in sizes of color sets), equipartite, and flat. The edge probability determines when the two vertices v_i and v_j are connected with an edge (v_i, v_j). This parameter was varied in the interval $p \in [0.004, 0.014]$ with a step of 0.0005. Thus, a phase transition phenomenon was captured, where graphs are hard to solve by most of the algorithms. Finally, the seeds were varied in the interval $q \in [1, 10]$ with a step of one. As a result, $3 \times 21 \times 10 = 630$ different graphs were obtained. In summary, each algorithm was executed 15,750 times to complete this experimental setup.

Phase transition is a phenomenon that accompanies almost all NP-hard problems and determines the region where the NP-hard problem passes over the state of "solvability" to a state of "unsolvability," and vice versa (Turner, 1988). Typically, this region is characterized by particular problem parameter. This parameter is the edge probability for 3-GCP. Many authors have determined this region differently. For example, Petford and Welsh (1989) stated that this phenomenon occurs when $2pn/3 \approx 16/3$, Cheeseman et al. (1991) when $2m/n \approx 5.4$, Hayes (2003) when $m/n \approx 2.35$, and Eiben et al. (1998) when $7/n \leq p \leq 8/n$. In the presented case, the phase transition needed to be by $p \approx 0.008$ over Petford and Welsh and over Cheeseman, by $p \approx 0.007$ over Hayes, and $p \in [0.007, 0.008]$ over Eiben et al.

4.4.3.2 Influence of the Edge Probability

The influence of edge probability on the performance of the tested graph coloring algorithms was investigated during this experiment. The tested algorithms solved the test suite of graph, as represented in Section 4.4.3.1. This test suite was selected so that the behavior of the graph coloring algorithms in the phase transition could be observed.

The results of this experiment are illustrated in Figure 4.3 and are divided into six diagrams corresponding to the graphs of different types (uniform, equipartite, and flat). Furthermore, the graphs are compared according to the measures SR and AES. In these diagrams, the average values of the corresponding measures are presented as obtained after 25 runs for each of 10 different seeds.

As shown in Figure 4.4(a), (c), and (e), the performance of MSA-FFA was similar to those of HEA and Tabucol when solving the instances of graphs lower than $p < 0.007$. With increasing the edge probability, performances of this algorithm within the phase transition region $p \in [0.007, 0.010]$ became worse, especially when solving the flat type of graphs. On the other hand, EA-SAW and FFA reported the worse results. When these two algorithms were compared with each other, it can be observed that EA-SAW gets stuck before FFA (e.g., when $p < 0.006$ for uniform and equipartite graphs) and improves the results faster than FFA (e.g., when $p > 0.01$ for the same types of graphs).

According to efficiency (measure AES in Figure 4.4(b), (d), and (f)), the best results were obtained by HEA. The performances of Tabucol were comparable especially when solving the flat graphs. MSA-FFA was competitive with these two algorithms when solving the uniform and equipartite graphs, while EA-SAW and FFA, on average, exposed the worst results for all three types of graphs. Interestingly, Tabucol and HEA also increased AES by $p = 0.013$. This behavior is connected with the phenomenon of second phase transition (Boettcher and Percus, 2004).

4.4.3.3 Influence of the Fitness Diversity Metric

The goal of this experiment was to show how exploration and exploitation are balanced by MSA-FFA. In line with this, the fitness diversity metric Ψ was measured in each generation on the phenotype level. The metric Ψ determines how exploration and exploitation are balanced by firefly search. In order to analyze how this measure behaves over the particular instance of a graph, all 25 runs of MSA-FFA were taken into consideration. Here, the equipartite graph with $p = 0.008$ and $q = 5$ was observed. Note that this instance of a graph is in the phase transition.

The results of this experiment are presented in (Figure 4.5), which is divided in two diagrams: diagram 4.5(a) represents the successfully finished runs (10 runs), while diagram 4.5(b) illustrates the unsuccessful runs (15 runs). In this case, the MSA-FFA reached SR = 0.40 by solving this instance of a graph.

According to Figure 4.5, the fitness diversity measure was stabilized by $\Psi \approx 0.2$. As a result, in each generation, about 30% of the new population was taken from the subpopulation $P_{LOW}^{(t)}$. Note that no significant differences were observed among the runs where the solution was found and runs where the solution was not found.

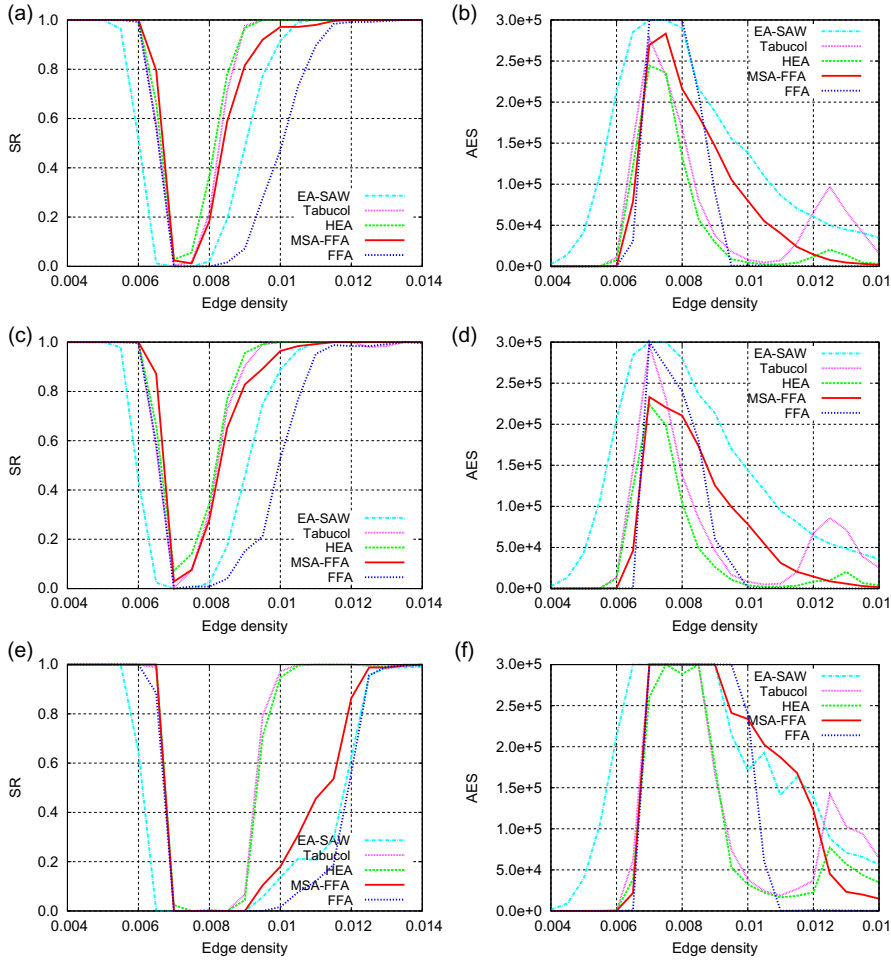

Figure 4.4 Influence of the edge probability on large-scale 3-colored graphs. (a) SR on uniform large-scale graphs. (b) AES on uniform large-scale graphs. (c) SR on equipartite large-scale graphs. (d) AES on equipartite large-scale graphs. (e) SR on flat large-scale graphs. (f) AES on flat large-scale graphs.

4.4.3.4 *Influence of the Inertia Diversity Metric*

The aim of this experiment was to investigate the behavior of the inertia diversity metric I_C. In contrast to fitness diversity Ψ, it is measured at the genotype level and describes how individuals are dispersed around the centroid (Eq. (4.13)). The theory of EAs for swarm intelligence (Neri, 2012) asserts that population diversity is high at the beginning of the search. As the algorithm progresses toward the better

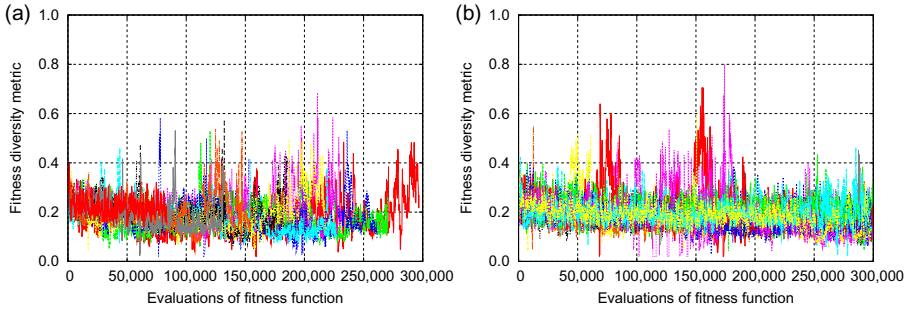

Figure 4.5 Influence of fitness diversity metric. (a) Solutions found. (b) Solutions not found.

regions of the search space, the population diversity decreases until either a solution is found or the algorithm gets stuck into local optimum. When all individuals in the population converge to the same point within fitness landscape, the inertia diversity metric decreases to zero. In order to prevent premature convergence, a new population model was applied to MSA-FFA. The experiment was conducted as follows: 25 runs of MSA-FFA were observed when coloring the equipartite graph with $p = 0.008$ and $q = 5$.

The results of this experiment are presented in Figure 4.6, which is divided into two diagrams. The former (diagram 4.6(a)) represents those runs where the solution was found, while the latter (diagram 4.6(b)) the runs where the solution was not found.

It can be seen from diagrams that at the beginning the population diversity measured the inertia diversity measure was high, but this diversity was lost by the population very quickly. Although during some runs the inertia diversity was near to zero, it never reached this value. That is, the new population model conducting the underfit individuals into population never gets stuck in local optimum. For instance, when using this population model, the SR on the mentioned instance of the graph was increased from $SR = 0.0$ by MFFA to $SR = 0.40$ by MSA-FFA.

4.4.3.5 Convergence Graphs

The convergence of solutions was studied during the last experiments, where the average fitness of the population in certain generation was compared with the best fitness found so far. Thus, two different runs of MSA-FFA were analyzed by solving the equipartite graph with $p = 0.008$ and $q = 5$: in the first, where the solution was found, while in the second, where the solution was not found.

The results of both runs can be seen in Figure 4.7, which is divided into diagrams 4.7(a) (solution found) and 4.7(b) (solution not found).

It can be seen from diagram 4.7(a) that the average fitness consists of a sequence of values that represents hills and valleys. The hills denote an increase of

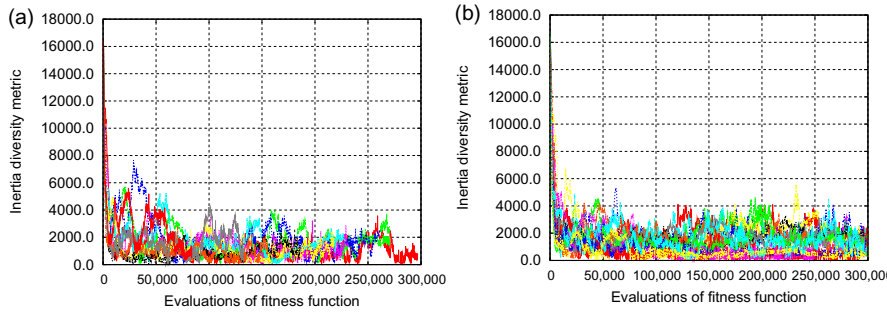

Figure 4.6 Influence of fitness diversity metric. (a) Solutions found. (b) Solutions not found.

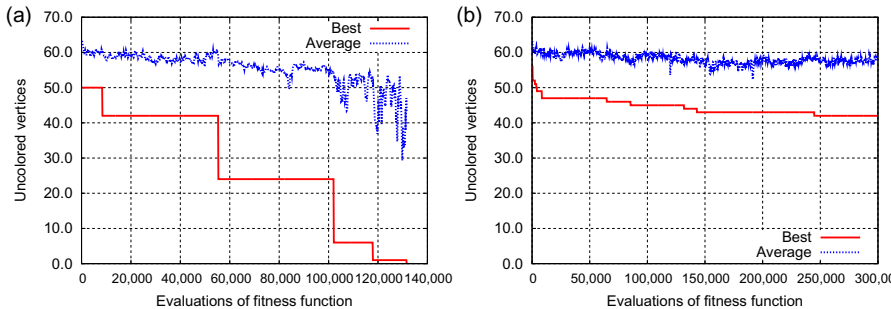

Figure 4.7 Convergence diagrams. (a) Solution found. (b) Solution not found.

the average fitness, while the valleys a decrease of it. However, the average fitness decreased slowly throughout the entire run. This decreasing was more observable at the end of the optimization. On the other hand, the best solution was decreased stepwise. After some big skips, the optimal solution was found.

When the solution was not found (diagram 4.7(b)), the average fitness remained almost constant during the whole run, while the best solution decreased stepwise. Unfortunately, these skips were smaller and usually led the search process to the local optimum.

4.4.3.6 Discussion

The results of these experiments can be summarized as follows: The results of MSA-FFA are comparable with Tabucol and HEA when solving 3-GCP on uniform and equipartite graphs and are slightly worse on flat graphs. However, EA-SAW and FFA gained the unsatisfactory results that are incomparable with the other tested algorithms.

The fitness diversity Ψ balances the exploration and exploitation within the firefly search algorithm. In our opinion, this metric plays a crucial role during optimization and we are convinced that its value depends on the problem to be solved. Although this metric behaved as a relative constant in the case of 3-GCP, we believe that it should be dependent on the number of fitness evaluations. That is, at the beginning of the optimization process, when the firefly search process ruthlessly exploits the solutions in the current population, some mechanism is necessary to prevent the loss of population diversity being too fast. In matured generations, when the diversity of the population is low, the exploration of the search space must be activated in order to prevent the search process to get stuck into local optimum. Both demands are being considered by the construction of a new population model.

The inertia diversity I_C, on the other hand, affirmed the thesis that the population diversity is high at the beginning of the optimization and becomes lower as the population becomes matured. Unfortunately, the population diversity by MSA-FFA is lost in the first 2% of the allowable number of fitness evaluations. Without using the new population model, MSA-FFA is unable to prevent the search process from premature convergence.

The convergence of the MSA-FFA to the global optimum is impossible to predict, as illustrated within convergence graphs. Population diversity is necessary in order to reach the optimum, but the population diversity does not ensure that the best solution is also found during the search. In summary, the population diversity is a prerequisite for convergence, but not also a sufficient condition.

4.5 Conclusions

The FFA is a member of the swarm intelligence algorithms inspired by the collective behavior of social insects and other animal societies when solving diverse optimization problems. Essentially, this algorithm has not extensively been applied to the domain of combinatorial optimization problems. Therefore, this work is among the first of its kind that focused on the behavior of the FFA when solving this class of problems.

In order to prepare the FFA algorithm for solving the combinatorial optimization problems, the following features were proposed:

- new population model,
- explicit control of exploration and exploitation during firefly search,
- self-adaptation of control parameters,
- incorporating local search heuristics.

The new population model introduces selection pressure within the firefly search process. Without this feature, the FFA algorithm is unable to direct deep searching into the promising regions of the search space. Explicit balancing between exploration and exploitation during the firefly search is realized using fitness diversity metric that is calculated in each generation. The control parameters are not set fix

during the search process but are self-adapted within the proposed algorithm. Local search heuristics are applied in order to incorporate problem-specific knowledge.

The proposed algorithm is named the MSA-FFA because of the characteristics of the used features. In order to show its quality, the MSA-FFA was applied to graph 3-coloring that is a well-known combinatorial optimization problem. Extensive experiments of MSA-FFA by coloring the large-scale graphs of various types, edge probabilities, and the seeds of a random graph generator were performed and compared with other well-known graph coloring algorithms, like Tabucol, HEA, and EA-SAW. The experiments showed that the results of MSA-FFA are comparable with the results obtained by the other algorithms used in experiments.

In the future, MSA-FFA should also be used for solving other combinatorial optimization problems. How the proposed hybridizations influence on the results of an FFA should be performed as well. Essentially, additional experiments with fitness diversity metric should be conducted in order to obtain more direct control over exploration and exploitation in the FFA, and in fact, in many other swarm-based algorithms as well.

References

Aarts, E., Lenstra, J., 1997. Local Search in Combinatorial Optimization. Princetown University Press, Princetown and Oxford.

Avanthay, C., Hertz, A., Zufferey, N., 2003. A variable neighborhood search for graph coloring. Eur. J. Oper. Res. 151, 379−388.

Bäck, T., 1998. An overview of parameter control methods by self-adaptation in evolutionary algorithms. Fundam. Informaticae. 35, 51−66.

Beni, G., Wang, J., 1989. Swarm intelligence in cellular robotic systems. Proceedings of the NATO Advanced Workshop on Robots and Biological Systems. Tuscany, Italy.

Blöchliger, I., Zufferey, N., 2003. A reactive tabu search using partial solutions for the graph coloring problem. In: Kral, D., Sgall, J. (Eds.), Coloring Graphs from Lists with Bounded Size of Their Union: Result from Dagstuhl Seminar 03391. Department of Applied Mathematics and Institute for Theoretical Computer Science, Prague.

Blöchliger, I., Zufferey, N., 2008. A graph coloring heuristic using partial solutions and a reactive tabu scheme. Comput. Oper. Res. 35 (3), 960−975.

Blum, C., Li, X., 2008. Swarm intelligence in optimization. In: Blum, C., Merkle, D. (Eds.), Swarm Intelligence: Introduction and Applications. Springer-Verlag, Berlin, pp. 43−86.

Blum, C., Roli, A., 2003. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. 35 (3), 268−308.

Blum, C., Puchinger, J., Raidl, G., Roli, A., 2011. Hybrid metaheuristics in combinatorial optimization: a survey. Appl. Soft Comput. 11 (6), 4135−4151.

Boettcher, S., Percus, A., 2004. Extremal optimization at the phase transition of the three-coloring problem. Phys. Rev. 69 (6), 66−73.

Bondy, J., Murty, U., 2008. Graph Theory. Springer-Verlag, Berlin.

Brelaz, D., 1979. New methods to color vertices of a graph. Commun. ACM. 22, 251−256.

Chams, M., Hertz, A., de Werra, D., 1987. Some experiments with simulated annealing for coloring graphs. Eur. J. Oper. Res. 32, 260−266.

Cheeseman, P., Kanefsky, B., Taylor, W., 1991. Where the really hard problems are. Proceedings of the International Joint Conference on Artificial Intelligence. Morgan Kaufmann, San Francisco, CA, pp. 331−337.

Chiarandini, M., Stützle, T., 2002. An application of iterated local search to graph coloring. In: Johnson, D., Mehrotra, A., Trick, M. (Eds.), Proceedings of the Computational Symposium on Graph Coloring and its Generalizations. Ithaca, New York, NY, pp. 112−125.

Chiarandini, M., Stützle, T., 2012. Online Compendium to the Article: An Analysis of Heuristics for Vertex Colouring. [Online] Available from: <http://www.imada.sdu.dk/~marco/gcp-study/> (accessed 20.09.12.).

Chiarandini, M., Dumitrescu, I., Stützle, T., 2007. Stochastic local search algorithms for the graph colouring problem. In: Gonzalez, T. (Ed.), Handbook of Approximation Algorithms and Metaheuristics, Computer and Information Science Series. Chapman & Hall/CRC, Boca Raton, FL, pp. 63.1−63.17.

Culberson, J., 2012. Graph Coloring Page. [Online] Available from: <http://www.ncbi.nlm.nih.gov/> (accessed 14.09.12.).

Črepinšek, M., Liu, S.-H., Mernik, M., 2011. Analysis of exploration and exploitation in evolutionary algorithms by ancestry trees. Int. J. Inn. Comput. Appl. 3 (1), 11−19.

Deb, K., 2001. Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley & Sons, Chichester, UK.

Dorigo, M., Di Caro, G., 1999. The ant colony optimization meta-heuristic. In: Corne, D., Dorigo, M., Glover, F. (Eds.), New Ideas in Optimization. McGraw Hill, London, UK, pp. 11−32.

Dorne, R., Hao, J., 1998. A new genetic local search algorithm for graph coloring. In: Eiben, A., Bäck, T., Schoenauer, M., Schwefel, H. (Eds.), Parallel Problem Solving from Nature—PPSN, Fifth International Conference, Lecture Notes in Computer Science. Springer-Verlag, Berlin, pp. 745−754.

Eiben, A., Smit, J., 2003. Introduction to Evolutionary Computing. Springer-Verlag, Berlin.

Eiben, A., van Der Hauw, J., van Hemert, J., 1998. Graph coloring with adaptive evolutionary algorithms. J. Heuristics. 4 (1), 25−46.

Eiben, A., Hinterding, R., Michalewicz, Z., 1999. Parameter control in evolutionary algorithms. IEEE Trans. Evol. Comput. 3 (2), 124−141.

Fister, I., Brest, J., 2011. Using differential evolution for the graph coloring. IEEE SSCI2011 Symposium Series on Computational Intelligence: Proceedings. IEEE, Piscataway, NJ, pp. 150−156.

Fister Jr., I., Fister, I., Brest, J., 2012a. A hybrid artificial bee colony algorithm for graph 3-coloring. Lecture Notes in Computer Science. Springer-Verlag, Berlin, pp. 66−74.

Fister Jr., I., Yang, X.-S., Fister, I., Brest, J., 2012b. Memetic firefly algorithm for combinatorial optimization. In: Filipič, B., Šilc, J. (Eds.), Bioinspired Optimization Methods and Their Applications: Proceedings of the Fifth International Conference on Bioinspired Optimization Methods and Their Applications—BIOMA 2012. Jožef Stefan Institute, Ljubljana, pp. 75−86.

Fister, I., Mernik, M., Filipič, B., 2013. Graph 3-coloring with a hybrid self-adaptive evolutionary algorithm. Comput. Optimiz. Appl. 54 (3), 741−770.

Fleurent, C., Ferland, J., 1996. Genetic and hybrid algorithms for graph coloring. Ann. Oper. Res. 63, 437−464.

Galinier, P., Hao, J., 1999. Hybrid evolutionary algorithms for graph coloring. J. Comb. Optimiz. 3 (4), 379−397.

Galinier, P., Hertz, A., 2006. A survey of local search methods for graph coloring. Comput. Oper. Res. 33, 2547−2562.
Galinier, P., Hertz, A., Zufferey, N., 2008. An adaptive memory algorithm for the *k*-coloring problem. Discrete Appl. Math. 156 (2), 267−279.
Gandomi, A., Yang, X.S., Alavi, A., 2011. Mixed variable structural optimization using firefly algorithm. Comput. Struct. 89 (23), 2325−2336.
Gandomi, A., Yang, X.S., Talatahari, S., Alavi, A., 2013. Firefly algorithm with chaos. Commun. Nonlin. Sci. Numer. Simul. 18 (1), 89−98.
Garey, M., Johnson, D., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., New York, NY.
Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. 13 (5), 533−549.
Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA.
Hayes, B., 2003. On the threshold. Am. Sci. 91, 12−17.
Hernández, H., Blum, C., 2012. Distributed graph coloring: an approach based on the calling behavior of Japanese tree frogs. Swarm Intell. 6 (2), 117−150.
Hertz, A., de Werra, D., 1987. Using tabu search techniques for graph coloring. Computing. 39, 345−351.
Hertz, A., Plumettaz, M., Zufferey, N., 2008. Variable space search for graph coloring. Discrete Appl. Math. 156 (13), 2551−2560.
Holland, J., 1992. Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, USA.
Hoos, H., Stützle, T., 2005. Stochastic Local Search. Foundations and Applications. Elsevier, Oxford.
Johnson, D., Aragon, C., McGeoch, L., Schevon, C., 1991. Optimization by simulated annealing: an experimental evaluation, part II; graph coloring and number partitioning. Oper. Res. 39 (3), 378−406.
Karaboga, D., Basturk, B., 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J. Global Optimiz. 39 (3), 459−471.
Kennedy, J., Eberhart, R., 1999. The particle swarm optimization: social adaptation in information processing. In: Corne, D., Dorigo, M., Glover, F. (Eds.), New Ideas in Optimization. McGraw Hill, London, UK, pp. 379−387.
Korošec, P., Šilc, J., Filipič, B., 2012. The differential ant-stigmergy algorithm. Inf. Sci. 192, 82−97.
Kubale, M., 2004. Graph Colorings. American Mathemetical Society, Rhode Island.
Lü, Z., Hao, J., 2010. A memetic algorithm for graph coloring. Eur. J. Oper. Res. 203, 241−250.
Malaguti, E., Toth, P., 2009. A survey on vertex coloring problems. Int. Trans. Oper. Res.1−34.
Malaguti, E., Monaci, M., Toth, P., 2008. A metaheuristic approach for the vertex coloring problem. INFORMS J. Comput. 20, 302−316.
Meyer-Nieberg, S., Beyer, H.-G., 2007. Self-adaptation in evolutionary algorithms. In: Lobo, F., Lima, C., Michalewicz, Z. (Eds.), Parameter Setting in Evolutionary Algorithms. Springer-Verlag, Berlin, pp. 47−76.
Morrison, R., 2004. Designing Evolutionary Algorithms for Dynamic Environments. Springer-Verlag, Berlin.
Moscato, P., 1999. Memetic algorithms: a short introduction. In: Corne, D., Dorigo, M., Glover, F. (Eds.), New Ideas in Optimization. McGraw Hill, London, UK, pp. 219−234.

Neri, F., 2012. Diversity management in memetic algorithms. In: Neri, F., Cotta, C., Moscato, P. (Eds.), Handbook of Memetic Algorithms, Studies in Computational Inteligence. Springer-Verlag, Berlin, pp. 153−165.

Neri, F., Toivanen, J., Cascella, G., Ong, Y.-S., 2007. An adaptive multimeme algorithm for designing HIV multidrug therapies. IEEE/ACM Trans. Comput. Biol. Bioinform. 4 (2), 264−278.

Petford, A., Welsh, D., 1989. A randomized 3-coloring algorithms. Discrete Math. 74 (1−2), 253−261.

Prügel-Bennett, A., 2010. Benefits of a population: five mechanisms that advantage population-based algorithms. IEEE Trans. Evol. Comput. 14 (4), 500−517.

Rechenberg, I., 1973. Evolutions strategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Fromman-Holzboog, Stuttgart.

Stadler, P., 1995. Towards a theory of landscapes. In: Lopez-Pena, R., et al., (Eds.), Lecture Notes in Physics: Complex Systems and Binary Networks. Springer-Verlag, Berlin, pp. 77−163.

Talatahari, S., Gandomi, A., Yun, G., 2012. Optimum design of tower structures using firefly algorithm. In: The Structural Design of Tall and Special Buildings. [Online] Available from: <http://dx.doi.org/10.1002/tal.1043/> (accessed 10.03.13).

Turner, J., 1988. Almost all *k*-colorable graphs are easy to color. J. Algorithms. 9 (1), 63−82.

van Hemert, J., 2012. Jano's Homepage. [Online] Available from: <http://www.vanhemert.co.uk/csp-ea.html/> (accessed 20.09.12.).

Wolpert, D., Macready, W., 1997. No free lunch theorems for optimization. IEEE Trans. Evol. Comput. 1, 67−82.

Wright, S., 1932. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In: Proceedings of the Sixth International Congress of Genetics. 1, 356−366.

Yang, X.S., 2008. Nature-Inspired Metaheuristic Algorithms. Luniver Press, Cambridge.

Yang, X.S., 2010. A new metaheuristic bat-inspired algorithm. In: Gonzalez, J., et al., (Eds.), Nature Inspired Cooperative Strategies for Optimization (NISCO 2010). Studies in Computational Inteligence, Springer-Verlag, Berlin, pp. 65−74.

Yang, X.S., Deb, S., 2009. Cuckoo search via Levy flights. World Congress on Nature and Biologically Inspired Computing (NaBIC 2009). IEEE, Coimbatore, India, pp. 210−214, (s.l.).

Yang, X.S., Hosseini, S., Gandomi, A., 2012. Firefly algorithm for solving non-convex economic dispatch problems with valve loading effect. Appl. Soft Comput. 12 (3), 1180−1186.