MEMETIC FIREFLY ALGORITHM FOR COMBINATORIAL OPTIMIZATION

Iztok Fister Jr., Iztok Fister, Janez Brest
Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia
iztok.fister@guest.arnes.si; {iztok.fister; janez.brest}@uni-mb.si

Xin-She Yang
National Physical Lab, Teddington, London, UK
xin-she.yang@npl.co.uk

Abstract Firefly algorithms belong to modern meta-heuristic algorithms inspired by nature that can be successfully applied to continuous optimization problems. In this paper, we have been applied the firefly algorithm, hybridized with local search heuristic, to combinatorial optimization problems, where we use graph 3-coloring problems as test benchmarks. The results of the proposed memetic firefly algorithm (MFFA) were compared with the results of the Hybrid Evolutionary Algorithm (HEA), Tabucol, and the evolutionary algorithm with SAW method (EA-SAW) by coloring the suite of medium-scaled random graphs (graphs with 500 vertices) generated using the Culberson random graph generator. The results of firefly algorithm were very promising and showed a potential that this algorithm could successfully be applied in near future to the other combinatorial optimization problems as well.

Keywords: Firefly algorithm, Graph 3-coloring, Nature inspired algorithms, NP-complete problem, Swarm intelligence.

1. Introduction

Nature, especially biological systems, has always been an inspiration for those scientists who would like to transform some successful features of a biological system into computer algorithms for efficient problem solving. Birds, insects, ants, and fish may display some so-called, collective or swarm intelligence in foraging, defending, path finding, etc. This collective intelligence of self-organizing systems or agents has served as a basis for many good and efficient algorithms developed in the past,

e.g.: ant-colony optimization [7], particle swarm optimization [22], artificial bee colony [11, 12, 21], bacterial foraging [23]. Today, all these algorithms are referred to as *swarm intelligence*.

The firefly algorithm (FFA) belongs to swarm intelligence as well. It was developed by X. S. Yang [28]. This algorithm is based on the behavior of fireflies. Each firefly flashes its lights with some brightness. This light attracts other fireflies within the neighborhood. On the other hand, this attractiveness depends on the distance between the two fireflies. The closer the two fireflies are, the more attractive they will seem to other. In FFA, each firefly represents a point in a search space. When the attractiveness is proportional to the objective function the search space is explored by moving the fireflies towards more attractive neighbors.

FFA has displayed promising results when applied to continuous optimization problems [29, 30]. Conversely, within the area of combinatorial optimization problems, only a few papers have been published to date. Therefore, aim of this paper is to show that FFA can be applied to this kind of optimization problems as well. In this context, FFA for graph 3-coloring (3-GCP) has been developed. 3-GCP can informally be defined as follows: How to color a graph G with three colors so that none of the vertices connected with an edge is colored with the same color. The problem is NP-complete as proved by Garey and Johnson [16].

The most natural way to solve this problem is in a greedy fashion. Here, the vertices are ordered into a permutation and colored sequentially one after the other. However, the quality of coloring depends on the order in which the vertices will be colored. For example, the *naive* method orders the vertices of graph randomly. One of the best traditional heuristics for graph coloring today is DSatur by Brelaz [2], which orders the vertices v according to *saturation degree* $\rho(v)$. The saturation degree denotes the number of distinctly colored vertices to the vertex v.

This problem cannot be solved by an exact algorithm for graph instances of more than 100 vertices. Therefore, many heuristic methods have been developed for larger instances of graphs. These methods can be divided into local search [15] and hybrid algorithms [26]. One of the more successful local search heuristic was Tabucol developed by Herz and De Werra [19], who employed the tabu search proposed by Glover [17]. The most effective local search algorithms today are based on reactive partial local search [1, 25], adaptive memory [20], and variable search space [1]. On the other hand, various evolutionary algorithms have been hybridized using these local search heuristics. Let us refer to three such algorithms only: the hybrid genetic algorithm by Fleurent and Ferland [13], the hybrid evolutionary algorithm by Galinier and Hao [14], and the memetic algorithm for graph coloring by Lü and Hao [24].

Some modifications of the original algorithm need to be performed in order to apply FFA to 3-GCP. The original FFA algorithm operates on real-valued vectors. On the other hand, the most traditional heuristics act on the permutation of vertices. In order to incorporate the benefits of both, solutions of the proposed memetic FFA (MFFA) algorithm are represented as real-valued vectors. The elements of these vectors represent *weigths* that determine how hard the vertices are to color. The higher the weight is, the sooner the vertex should be colored. The permutation of vertices is obtained by sorting the vertices according to their weights. The DSatur traditional heuristic is used for construction of 3-coloring from this permutation. A similar approach was used in the evolutionary algorithm with the SAW method (EA-SAW) of Eiben et al. [8], and by the hybrid self-adaptive differential evolution and hybrid artificial bee colony algorithm of Fister et al. [10, 12]. Additionally, the *heuristical swap* local search is incorporated into the proposed MFFA. In order to preserve the current best solution in the population, the elitism is considered by this algorithm.

The results of the proposed MFFA algorithm for 3-GCP were compared with the results obtained with EA-SAW, Tabucol, and HEA by solving an extensive set of random medium-scale graphs generated by the Culberson graph generator [6]. The comparison between these algorithms shows that the results of the proposed MFFA algorithm are comparable, if not better, than the results of other algorithms used in the experiments.

The structure of this paper is as follows: In Section 2, the 3-GCP is discussed, in detail. The MFFA is described in Section 3, while the experiments and results are presented in Section 4. The paper is concluded with a discussion about the quality of the results, and the directions for further work are outlined.

2. Graph 3-Coloring

Let us suppose, an undirect 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 $i = 1, \ldots, m$ to the unordered pair $e = \{v_i, v_j\}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, n$. Then, the vertex 3-coloring (3-GCP) is defined as a mapping $c : V \rightarrow S$, where $S = \{1, 2, 3\}$ is a set of three colors and c 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 are colored with a different color.

3-GCP can be formally defined as a constraint satisfaction problem (CSP). It is represented as a pair $\langle S, \phi \rangle$, where S denotes the search

space, in which all solutions $\mathbf{s} \in S$ are *feasible*, and ϕ a Boolean function on S (also a *feasibility condition*) that divides the search space into *feasible* and *unfeasible* regions. To each $e \in E$ the constraint b_e is assigned with $b_e(\langle s_1, \ldots, s_n \rangle) = true$ 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 farther the unfeasible solution is from the feasible region, the higher is the penalty function. The penalty function is expressed as:

$$f(\mathbf{s}) = min \sum_{i=0}^{n} \psi(\mathbf{s}, B^i), \tag{1}$$

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

$$\psi(\mathbf{s}, B^i) = \begin{cases} 1 & \text{if } \mathbf{s} \text{ violates at least one } b_e \in B^i, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Note that Eq. (1) also represents the objective function. On the other hand, the same equation can be used as a feasibility condition in the sense that $\phi(\mathbf{s}) = true$ if and only if $f(\mathbf{s}) = 0$. If this condition is satisfied a proper graph 3-coloring is found.

3. Memetic Firefly Algorithm for Graph 3-Coloring

The phenomenon of fireflies is that fireflies flash their lights that can be admired on clear summer nights. This light is produced by a complicated set of chemical reactions. Firefly flashes in order to attract mating partners and serve as a protection mechanism for warning off potential predators. Their light intensity I decreases when the distance r from the light source increases according to term $I \propto r^2$. On the other hand, air absorbs the light as the distance from the source increases.

When the flashing light is proportional to the objective function of the problem being optimized (i.e., $I(\mathbf{w}) \propto f(\mathbf{w})$), where $\mathbf{w}$ represents the candidate solution) this behavior of fireflies can represent the base for an optimization algorithm. However, artificial fireflies obey the following rules: all fireflies are unisex, their attractiveness is proportional to their brightness, and the brightness of a firefly is affected or determined by the landscape of the objective function.

These rules represent the basis on which the firefly algorithm acts [28]. The FFA algorithm is population-based, where each solution denotes a

point in the search space. The proposed MFFA algorithm is hybridized with a local search heuristic. In this algorithm, the solution is represented as a real-valued vector $\mathbf{w_i} = (w_{i,1}, \ldots, w_{i,n})$ for $i = 1 \ldots NP$, where NP denotes the size of population P. The vector $\mathbf{w_i}$ determines the weights assigned to the corresponding vertices. The values of the weights are taken from the interval $w_{i,j} \in [lb, ub]$, where lb and ub are the lower and upper bounds, respectively. The weights represent an initial permutation of vertices $\pi(\mathbf{v_i})$. This permutation serves as an input to the DSatur heuristic that obtains the graph 3-coloring. The pseudocode of the MFFA algorithm is illustrated in Algorithm 1.

Algorithm 1 Pseudo code of the MFFA algorithm

```
1: t = 0; fe = 0; found = FALSE; s* = ∅;
2: P^(t) = InitializeFFA();
3: while (!TerminateFFA(fe, found)) do
4:     fe += EvaluateFFA(P^(t));
5:     P' = OrderFFA(P^(t));
6:     found = FindTheBestFFA(P^(t), s*);
7:     P^(t+1) = MoveFFA(P^(t), P');
8:     t = t+1;
9: end while
```

As can be seen from Algorithm 1, the search process of the MFFA algorithm (statements within the **while** loop) that is controlled by generation counter t consists of the following functions:

- EvaluateFFA(): evaluating the solution. This evaluation is divided into two parts: In the first part, the solution $\mathbf{w_i}$ is transformed into a permutation of vertices $\pi(\mathbf{v_i})$ according to the non-decreasing values of the weights. In the second part, the permutation of vertices $\pi(\mathbf{v_i})$ is decoded into a 3-coloring $\mathbf{s}_i$ by the DSatur heuristic. Note that the 3-coloring $\mathbf{s}_i$ represents the solution of 3-GCP in its original problem space, where the quality of the solution is evaluated according to Eq. (1). Conversely, looking for a new solution is performed within the real-valued search space.

- OrderFFA(): forming an intermediate population P' by copying the solutions from the original population $P^{(t)}$ and sorting $P^{(t)}$ according to the non-decreasing values of the objective function.

- FindTheBestFFA(): determining the best solution in the population $P^{(t)}$. If the best solution in $P^{(t)}$ is worse than the $\mathbf{s}^*$ the later replaces the best solution in $P^{(t)}$, otherwise the former becomes the best solution found so far $\mathbf{s}^*$.

- MoveFFA(): moving the fireflies towards the search space according to the attractiveness of their neighbour's solutions.

Two features need to be developed before this search process can take place: the initialization (function InitializeFFA()) and termination (function TerminateFFA()). The population is initialized randomly according to the following equation:

$$w_{i,j} = (ub - lb) \cdot rand(0,1) + lb, \tag{3}$$

where the function rand$(0,1)$ denotes the random value from the interval $[0,1]$. The process is terminated when the first of the following two condition is satisfied: the number of objective function evaluations *fe* reaches the maximum number of objective function evaluations (MAX_FES) or the proper coloring is found ($found == true$).

The movement of i-th firefly is attracted to another more attractive firefly j, and expressed as follows:

$$\mathbf{w_i} = \mathbf{w_i} + \beta_0 e^{-\gamma r_{i,j}^2}(\mathbf{w_j} - \mathbf{w_i}) + \alpha(rand(0,1) - \frac{1}{2}), \text{ for } j = 1...n. \tag{4}$$

Note that the move of the i-th firefly is influenced by all the j-th fireflies for which $I[j] > I[i]$. As can be seen from Eq. (4), two summands are added to the current firefly position $\mathbf{w_i}$. The former reflects the attractiveness between firefly i and j (determined by $\beta(r) = \beta_0 e^{-\gamma r_{i,j}^2}$), while the latter is the randomized move in the search space (determined by the randomized parameter α). Furthermore, the attractiveness depends on the β_0 that is the attractiveness at $r = 0$, absorbtion coefficient γ, and Euclidian distance between the attracted and attracting firefly $r_{i,j}$.

3.1 The Heuristical Swap Local Search

After the evaluation step of the MFFA algorithm, the heuristical swap local search tries to improve the current solution. This heuristic is executed until the improvements are detected. The operation of this operator is illustrated in Fig. 1, which deals with a solution on G with 10 vertices. This solution is composed of a permutation of vertices $\mathbf{v}$, 3-coloring $\mathbf{s}$, weights $\mathbf{x}$, and saturation degrees ρ. The heuristical swap unary operator takes the first uncolored vertex in a solution and orders the predecessors according to the saturation degree descending. 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 (1-opt neighborhood).

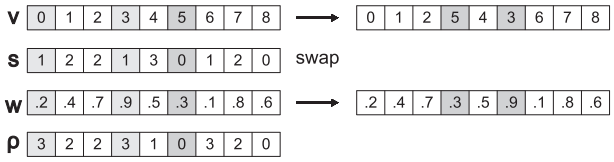


Figure 1. The heuristical swap unary operator.

In Fig. 1, 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 3 and swaps it with vertex 5 (the right-hand side of Fig. 1).

4. Results

In the experimental work, the results of the proposed MFFA algorithm were compared with the results of: EA-SAW, HEA, and Tabucol. The algorithms used in the experiments were not selected coincidentally. In order to help the developers of new graph coloring algorithms, the authors Chiarandini and Stützle [4] made the code of Tabucol and HEA available within an online compendium [5]. On the other hand, this study is based on the paper of Eiben and al. [8], in which the authors proposed the evolutionary algorithm with SAW method for 3-GCP. The source code of this algorithm can also be obtained from Internet [18]. The goal of this experimental work was see whether the MFFA could also be applied to combinatorial optimization problems like 3-GCP.

The characteristics of the MFFA in the experiments were as follows. The population size was set at 500. The MAX_FES was fixed at 300,000 by all algorithms to make this comparison as fair as possible. All algorithms executed each graph instance 25 times. The algorithm parameters of MFFA were set as follows: $\alpha = 0.1$, $\beta_0 = 0.1$, and $\gamma = 0.8$. Note that these values of algorithm parameters optimize the performance of MFFA and were obtained during parameter tuning within the extensive experimental work. This parameter tuning satisfies the first perspective of parameter tuning, as proposed by Eiben and Smit [9], i.e., choosing parameter values that optimize the algorithm's performance. In order to satisfy the second perspective of the parameter tuning, i.e., how the MFFA performance depends on its parameter values, only the influence of the edge density was examined because of limited paper length.

The algorithms were compared according to the measures *success rate* (SR) and *average evaluations of objective function to solution* (AES). While the first measure represents the ratio between the number of suc-

cessfully runs and all runs, the second determines the efficiency of a particular algorithm. The aim of these preliminary experiments was to show that MFFA could be applied for 3-GCP. Therefore, any comparison of algorithms according to the time complexity was omitted.

4.1 Test-Suite

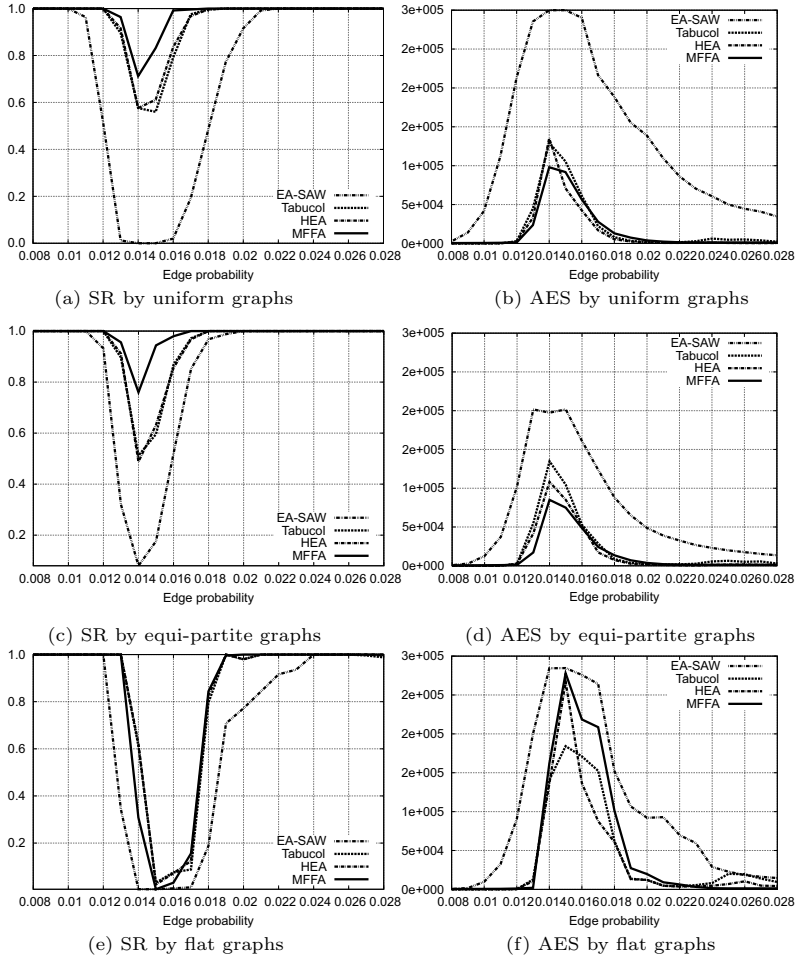
The test-suite, considered in the experiments, consisted of graphs generated with the Culberson random graph generator [6]. The graphs generated by this generator are distinguished according to type, number of vertices n, edge probability p, and seeds of random number generator q. Three types of graphs were employed in the experiments: *uniform* (random graphs without variability in sizes of color sets), *equi-partite*, and *flat*. The edge probabilities were varied in the interval $p \in 0.008 \ldots 0.028$ with a step of 0.001. Finally, the seeds were varied in interval $q \in 1 \ldots 10$ with a step of one. As a result, $3 \times 21 \times 10 = 630$ different graphs were obtained. That is, each algorithm was executed $15,750$ times to complete this experimental setup.

The experimental setup was selected so that a *phase transition* was captured. The 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 [31]. Typically, this region is characterized by ascertain problem parameter. This parameter is edge probability for 3-GCP. Many authors have determined this region differently. For example, Petford and Welsh [27] stated that this phenomenon occurs when $2pn/3 \approx 16/3$, Cheeseman et al. [3] when $2m/n \approx 5.4$, and Eiben et al. [8] when $7/n \leq p \leq 8/n$. In the presented case, the phase transition needed to be by $p = 0.016$ over Petford and Welsh, by $p \approx 0.016$ over Cheeseman, and between $0.014 \leq p \leq 0.016$ over Eiben et al.

4.2 Influence of the Edge Density

During this experiment, the influence of the edge density on the performance of the tested algorithms were investigated. The results are illustrated in Fig. 2. This figure consists of six diagrams corresponding to graphs of different types (uniform, equi-partite, and flat), and according to the measures SR and AES. In these diagrams, the average of those values accumulated after 25 runs are presented. Especially, we focus on the behavior of tested algorithms within the phase transition.

As can be seen in Fig. 2.a and Fig. 2.c, the results of MFFA outperformed the results of the other algorithms according to the measure SR on uniform and equi-partite graphs. HEA was slightly better than



(a) SR by uniform graphs (b) AES by uniform graphs

(c) SR by equi-partite graphs (d) AES by equi-partite graphs

(e) SR by flat graphs (f) AES by flat graphs

Figure 2. Results of algorithms for 3-GCP solving different types of random graphs.

Tabocol during the phase transition ($p \in [0.014, 0.016]$), whilst EA-SAW exposed the worst results within this region. As can be seen in Fig. 2.e, flat graphs were the hardest nuts to crack for all algorithms. Here, the results of MFFA according to measure SR were slightly worse but comparable to the results of HEA and Tabucol, whilst EA-SAW reported the worst results.

According to the measure AES (Fig. 2.b and Fig. 2.d), the best results were reported for MFFA by coloring the uniform and equi-partite graphs. On average, MFFA found solutions using a minimum number of evaluations. Note that the highest peak by $p = 0.014$ denotes the

hardest instance of uniform and equi-partite graphs to color for almost all the observed algorithms. Conversely, the graph with $p = 0.015$ was the hardest instance when coloring the flat graphs.

In summary, the proposed MFFA outperformed the results of HEA and Tabucol when coloring the uniform and equi-partite graphs, while by coloring the flat graphs it behaved slightly worse. The results of EA-SAW fell behind the results of the other tested algorithms for coloring all other types of graphs.

4.3 Discussion

The results of other parameter tuning experiments have been omitted because of limited paper length. Notwithstanding, almost four characteristics of MFFA could be exposed from the results of the last experiment. First, it is very important whether the movement of i-th firefly according to Eq. (4) is calculated from the position of the j-th firefly taken from the population $P^{(t)}$ or the intermediate population P', because the former incorporates an additional randomness within the search process. Thus, the results were significantly improved. Second, an exploration of the search space depends on the best solution in the population that directs the search process to more promising regions of the search space. As a result, this elitist solution needs to be preserved. Third, the local search serves as a search mechanism for detailed exploration of the basins of attraction. Consequently, those solutions that would normally be ignored by the regular FFA search process can be discovered. Fourth, the α parameter determines the size of the randomness move within the search space. Unfortunately, all conducted tests to self-adapt this parameter did not bear any improvement, at this moment. In summary, these preliminary results of MFFA encourage us to continue developing this algorithm in the future.

5. Conclusion

The results of MFFA showed that this algorithm could in future be a very promising tool for solving 3-GCP and, consequently, the other combinatorial optimization problems as well. In fact, it produced better results than HEA and Tabucol, when coloring the medium-scale uniform and equi-partite graphs. Unfortunately, this algorithm is slightly worse on flat graphs that remains the hardest to color for all the tested algorithms.

However, these good results could be misleading until further experiments on large-scale graphs (graphs with 1,000 vertices) are performed.

Fortunately, in the sense of preserving the obtained results, we have several ways for improving this MFFA in the future.

References

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

[2] D. Brelaz. New methods to color vertices of a graph. *Commun. ACM*, 22(4):251–256, 1979.

[3] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proc. International Joint Conference on Artificial Intelligence*, volume 1, pages 331–337, 1991.

[4] M. Chiarandini and T. Stützle. An aAnalysis of heuristics for vertex colouring. *Lect. Notes Comput. Sc.*, 6049:326–337, 2010.

[5] M. Chiandini and T. Stützle. Online compendium to the article: An Analysis of Heuristics for Vertex Colouring. `http://www.imada.sdu.dk/~marco/gcp-study/`. Accessed at 20 March 2012.

[6] J. Culberson. Graph Coloring Page. `http://www.ncbi.nlm.nih.gov`. Accessed at 20 March 2012.

[7] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[8] Á. E. Eiben, K. Hauw, and J. I. Hemert. Graph coloring with adaptive evolutionary algorithms. *J. Heuristics*, 4(1):25–46, 1998.

[9] Á. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[10] I. Fister and J. Brest. Using differential evolution for the graph coloring. In *Proc. IEEE Symposium Series on Computational Intelligence*, pages 150–156, 2011.

[11] I. Fister, I. Fister Jr., J. Brest, and V. Žumer. Memetic Artificial bee colony algorithm for large-scale global optimization. In *Proc. IEEE Congress on Evolutionary Computation*, 2012.

[12] I. Fister Jr., I. Fister, and J. Brest. A hybrid artificial bee colony algorithm for graph 3-coloring. In *Proc. 11th International Conference on Artificial Intelligence and Soft Computing* , 2012.

[13] C. Fleurent and J. Ferland. Genetic and hybrid algorithms for graph coloring. *Ann. Oper. Res.*, 63(3):437–464, 1996.

[14] P. Galinier and J. Hao. Hybrid evolutionary algorithms for graph coloring. *J. Comb. Optim.*, 3(4):379–397, 1999.

[15] P. Galinier and A. Hertz. A survey of local search methods for graph coloring. *Comput. Oper. Res.*, 33(9):2547–2562, 2006.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., 1979.

[17] F. Glover. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, 13(5):533–549, 1986.

[18] J. I. Van Hemert. Jano's Homepage. `http://www.vanhemert.co.uk/csp-ea.html`. Accessed at 20 March 2012.

[19] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

[20] A. Hertz, M. Plumettaz, and N. Zufferey. Variable Space Search for Graph Coloring. *Discrete Appl. Math.*, 156(13):2551–2560, 2008.

[21] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J. Global Optim.*, 39(3):459–471, 2007.

[22] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proc. IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[23] Y. Liu and K.M. Passino. Biomimicry of social foraging bacteria for distributed optimization: Models, principles, and emergent behaviors. *J. Optimiz. Theory App.*, 115(3):603–628, 2002.

[24] Z. Lü and J. K. Hao. A memetic algorithm for graph coloring. *Eur. J. Oper. Res.*, 203(1):241–250, 2010.

[25] E. Malaguti, M. Monaci and P. Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS J. Comput.*, 20(2):302–316, 2008.

[26] E. Malaguti and P. Toth. A survey on vertex coloring problems. *Intl. Trans. Oper. Res.*, 17(1):1–34, 2009.

[27] A. D. Petford and D. J. A. Welsh. A randomized 3-coloring algorithms. *Discrete Math.*, 74(1-2):253–261, 1989.

[28] X.-S. Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[29] X.-S. Yang. Firefly Algorithm, Lévy Flights and Global Optimization. In M. Bramer, R. Ellis, and M. Petridis, editors, Springer London, *Research and Development in Intelligent Systems*, volume 26, pages 209–218, 2010.

[30] X.-S. Yang. Firefly algorithm, stochastic test functions and design optimization. *Int. J. Bio-Inspired Computation*, 2(2):78–84, 2010.

[31] J. S. Turner. Almost all k-colorable graphs are easy to color. *J. Algorithm.*, 9(1):63–82, 1988.