Parallelization of the Firefly Algorithm for Unconstrained Optimization Problems

Milos SUBOTIC, Milan TUBA, Nadezda STANAREVIC
Faculty of Computer Science
University Megatrend Belgrade
Bulevar umetnosti 29
SERBIA
milos.subotic@gmail.com, tuba@ieee.org, srna@stanarevic.com

Abstract: - Bio-inspired metaheuristics for hard optimization problems are very commonly parallelized. In this paper a parallelization of firefly algorithm is introduced. Firefly algorithm is one of the latest bio-inspired algorithms and this is the first attempt of its parallelization. We used a set of well-known unconstrained continuous functions for benchmark purposes. Two approaches were implemented here - one aiming to faster execution and the other one aiming to better results. In the first approach, one run of the algorithm is presented as one thread, while in the second approach that we call multiple colony approach, two colonies of fireflies run simultaneously, and after certain number of generations the process of exchanging fireflies among them occurs. In this way we managed to obtain better results than the original algorithm.

Key-Words: - Firefly algorithm, Metaheuristics, Unconstrained optimization, Parallelization

1 Introduction

Optimization is certainly one of the most interesting scientific fields. Many methods have been developed for solving real-life optimization problems. These methods can roughly be divided into two groups [1]: rigorous and approximation methods.

Rigorous methods such as linear programming, integer programming, dynamic programming and branch-and-bound are usually applied to medium-size problems. For these kinds of problems they obtain satisfying results.

However, real-life optimization problems are often large scaled, and the process of finding optimal solution for these problems can be challenging for rigorous techniques. Optimization of such complex systems requires exponential amount of computing power and time with increase of number of decision variables. Approximation methods have been proposed for solving these kinds of problems [2].

Representative of approximation methods are nature inspired algorithms. As other methods from this group, they find near optimal solution in reasonable amount of computation time. Nature inspired algorithms adopted some of well-known features of natural systems which evolved though selection mechanism over millions of years. Two important features of natural systems which were incorporated into computer algorithms are selection of the fittest and adaptation to the surrounding. In nature inspired algorithms they manifest as [3]: intensification and diversification. Intensification is search process around current solutions in the population and it is known as exploitation. On the other hand, diversification maintains exploration of the search space and prevents algorithm of being stuck in local optimum.

Swarm intelligence algorithms are among the latest nature inspired metaheuristics. These are complex computational systems which mimic behavior of species such as ants, birds, fish, bees, cuckoos, frogs, etc. These animals (insects) have very finite individual capability, but when they act as a group (swarm), they can perform many complex tasks in order to survive. Using analogy with real swarm systems, swarm intelligence algorithms consist of many intelligent agents with complex interactions among them and without centralized supervision component. Each agent responds to local stimuli independently, but also, they all cooperate in order to accomplish global task.

Many swarm intelligence algorithms were developed and enhanced over the time. Particle swarm optimization (PSO) is well-known swarm intelligence algorithm which simulates collective behavior of fish or birds. It was developed by

This research is supported by Ministry of Education and Science, Republic of Serbia, Project No. III-44006

Eberhard and Kennedy [4] for optimizing nonlinear functions in multidimensional space.

Ant colony optimization (ACO) simulates foraging behavior of ants which use pheromone substance to find the shortest path between their nests and food source [4]. This algorithm, as well its modifications, found wide-spread use in many practical applications [6].

Karaboga was inspired by behavior of honey bee swarms and devised artificial bee colony (ABC) algorithm [7]. ABC algorithm uses three kinds of artificial bees: employed, onlooker and scout. Employed bees and onlookers are exploiting search space, while scout maintain diversification through exploration. There are also other approaches which model bee's behavior [8].

Cuckoo search (CS) algorithm is relatively new metaheuristics for continuous optimization problems [9]. CS uses Lévy flights search pattern and shows very promising performance.

Firefly (FF) algorithm is a novel metaheuristic which mimics flashing behavior of light bugs [10]. As the primary purpose of firefly's flash is to act as a signal system to attract other fireflies, researches have been attracted to incorporate such behavior into computation algorithm.

Very similar to FF algorithm is glow-worm swarm algorithm which has been used in many practical fields including robotics [11].

Besides above mentioned, there are also other swarm intelligence algorithms are often used with other evolutionary computation (EC) algorithms [12], and applied to wide variety of practical problems [12], [13].

This paper is devoted to FF algorithm. Our objective is to describe implementation of the parallelized FF algorithm and to examine how multithreaded algorithm's execution influences results. Parallelized FF algorithm was adapted for solving unconstrained continuous optimization problems. These optimization problems can be formulated as minimization or maximization of D-dimensional function:

$$\text{Min (or max) } f(x), \quad x=(x_1,x_2,x_3,\ldots x_D) \tag{1}$$

where D is the number of parameters to be optimized.

The search space is usually limited to n-dimensional hyper-rectangular space in R^n defined by lower and upper variables' bounds:

$$lb_i \le x_i \le ub_i, \quad 1 \le i \le n \tag{2}$$

This paper is organized as follows. After introduction in Section 1, detailed description of FF algorithm in Section 2 is introduced. Section 3 describes parallelization of the FF algorithm. In subsequent Section 4 testing results on standard unconstrained benchmark functions are presented. Finally, Section 5 includes conclusion and final remarks.

2 Firefly Algorithm

2.1. Fireflies in nature

Fireflies are also known as lighting bugs. Many people use terms firefly and glow-worm interchangeably. Both are beetles that can emit light and are mostly of the family of Lampyridae. But, there is substantial difference between glow-worms and fireflies – glow-worms belong to the group of flightless species of Lampyridae, while fireflies are recognized as species that have the ability to fly.

These insects can generate light inside of their bodies owing to a special chemical reaction. The process of generating light happens in unique organs in firefly's lower abdomen called light-emitters. Lighting system was used for sending warning signals among firefly population and to attract the potential victim, but now it has evolved that its primary purpose is in mate selection. It is said that these organisms use bioluminescence for sexual selection [14].

The pattern of flashes depends on particular species of fireflies. Both sexes of fireflies are brought together via the rhythmic flash, the rate of flashing and the amount of time form part of the signal system. Groups of fireflies have the ability to synchronize their flashes and this appearance is known as phase synchronization [14].

Male's unique pattern of flashing attracts female from the same species for mating. But, in some species, female can set up a decoy by mimicking the mating flashing pattern in order to eat males.

As the distance from the lighting source increases, the light intensity decreases. So, light intensity follows the inverse square law:

$$I \sim \frac{1}{r^2} \tag{3}$$

where I is light intensity, and r is distance. Besides that, the air also absorbs part of the light, and the light becomes even weaker.

Two mentioned factors affect the communicating abilities of fireflies. Residual light intensity is usually good enough for fireflies to communicate with each other.

These properties of lightening communication of fireflies can be translated into objective function to be optimized and that is exactly what makes FF algorithm possible to implement.

2.2. Algorithm Details

In firefly algorithm, the main goal of light signal system is to attract other fireflies. Whole system is complex and it must be simplified due to incorporation into evolutionary algorithm. The most important issues are flushing characteristic of fireflies which must be idealized.

There are three simplification rules which guide the construction of the FF algorithm. Rules are summarized in bullet points below:

- First rule emphasis that each firefly attracts all other fireflies with weaker flashes [15]. Therefore, all fireflies must be idealized in a way that they have only one sex and are attracted with each other.
- Second rule notes that attractiveness of fireflies is proportional to their brightness. On the other side, brightness is reverse proportional to the distance from the light source. That means that the less bright firefly will relocate towards the brighter one. If the particular firefly is the brightest one in the population, it will move in a random manner because there is no other bug to attract it.
- Finally, the third rule highlights that the brightness of a firefly is affected or determined by the distribution of the objective function.

According to rules presented above, pseudo code for the FF algorithm is:

```
Generate initial population of fireflies $x_i$ ($i$ =1,2…, $n$)
Light intensity $I_i$ at $x_i$ point is defined by $f(x)$
Define light absorption coefficient $\gamma$
Define numberOfGeneration
while (t < numberOfGeneration) do
    for (i = 1 to numberOfAllFireflies) do
        for (j = 1 to i) do
            If ($I_j < I_i$) then
                Move firefly j towards i in d dimension
                Attractiveness depends on distance r and absorption property of surroundings $\gamma$ via the function $exp^{-\gamma r}$.
                Evaluate new solutions and update light intensity
            end if
        end for
    end for
    Rank the fireflies, find the current best and move it in a random fashion
end while
```

For complete understanding of the algorithm few sentences about attractiveness of firefly and its movement should be added. First thing that comes to mind is to use an objective function *f(x)* to encode the brightness of a given firefly. Actually, it represents the light intensity at location *x* as *I(x)* = *f(x)*. Yet, there are some issues with the distance, point of view and the fact that the environment absorbs part of emitted light. At the source, the brightness is higher than at some distant point. Also, the brightness decreases while environment absorbs the light while it is travelling. It can be concluded that the attractiveness of firefly β is relative. It is known that the light intensity $I_{(r)}$ varies following the inverse square law:

$$I(r) = \frac{Io}{r^2} \tag{4}$$

where I_0 represents the light intensity at the source. If light absorption coefficient γ is added to Equation (4) it transforms to:

$$I(r) = \frac{Io}{1+\gamma r^2} \tag{5}$$

Constant 1 is added to denominator to avoid singularity of the term at the source ($r=0$).

Eq. 6 can also be used because of the fact that attractiveness β is proportional to intensity:

$$\beta\,(r) = \frac{\beta 0}{1+\gamma r^2} \tag{6}$$

Eq. 6 can be further approximated with Gaussian form [16] :

$$\beta\,(r) = \beta_0 \exp^{-\gamma r2} \tag{7}$$

Fireflies movement is based on the principles of attractiveness: when firefly *j* is more attractive than firefly *i* the movement is determined by the following equation:

$$x_i = x + \beta_0\, exp^{-\gamma r2}{}_{xi}\,(x_j - x_i) + \alpha\,(rand - \tfrac{1}{2}) \tag{8}$$

Third term is randomization term, where $\alpha \in [0,1]$ and *rand* is pseudo-random number between 0 and 1. Distance r_{ij} between fireflies *i* and *j* is calculated using Cartesian distance form:

$$r_{ij} = \sqrt{\textstyle\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2} \tag{9}$$

3 Parallelized FF Algorithm

We are witnessing a dramatic change in computer architecture due to the multicore paradigm shift. In general, a system of n parallel processors, each of speed k is less efficient than one processor of speed $n * k$. However, the parallel system is usually much cheaper to build and its power consumption is significantly lower.

Parallelization of algorithms has been proven to be a very powerful method in the case of population based algorithms [17]. There are several approaches in parallelization of bio-inspired heuristics, but they can be divided into two groups. Methods that are trying to reduce the execution time belong to the first group, while methods that are aiming on better results are in the second group. Here, one method from each group is applied to the FF algorithm and presented.

The methods that are trying to improve CPU utilization are commonly making function evaluation calls to run simultaneously. Some of them are running evaluation of every population individual as a separate thread. In most cases, this is not the best solution. Population based algorithm often have between 20 and 100 individuals in the colony. The evaluation function usually does not require much CPU time, so extensive use of CPU time for creating threads and their synchronization exceeds the benefits of parallel execution of each evaluation. Creating and synchronizing such large number of threads can be slower by far than using a serial execution of evaluations.

It is desirable to run population based heuristics many times, because they do not provide exact result but rather give approximation as final result. It is quite useful to run all iterations simultaneously in order to save time. This is most common method of parallelization of bio-inspired heuristics - one run of the algorithm is presented as one thread. In this approach threads have no communication between themselves at all. Every thread runs the same sequential FF algorithm with different random seeds. The final solution is the best one of all the independent runs. The speed increases almost as many times as there are execution cores in system. Independent parallel runs approach is too coarse grained and there are no speed gains for one single runtime. On single core system this implementation can be slower than serial execution of all runs. This can be explained by high cost of switching CPU between threads.

Methods that are aiming for better results are usually dividing population into certain number of sub-populations and then exchanging the results among them after certain number of generations. The idea was to use more than one colony on same search space. Multiple colonies can find more useful solutions and narrow the search space. Trapping in local optimum can be avoided by using numerous colonies.

In this paper, the emphasis is on multiple colonies approach where the question of number of colonies arises. In our tests we empirically found that best results are obtained when the main colony of fireflies is divided into two sub-colonies. There is also an issue of frequency of communication between two colonies. The period between two communications can be determined by the number of cycles or by the time unit. In our experience, it is better to use the number of cycles rather than time unit. Since on different systems various amounts of computational work can be done in the same time, only a few communications can occur on the faster systems, while during the same algorithm execution on the slower system, number of communications can be significantly greater.

When algorithm starts execution, after certain number of cycles a new population is created. According to our experiments, the best results are obtained when the creation of new population is done after every 1000 cycles. New population is created in the following way: half of individuals in new population are copied from one sub-colony and half from the other. Before creating new population, the fireflies are sorted by their light intensity, and then the half with greater light intensity is copied into new population. As a result, new population contains best fireflies from both sub-colonies. After that, sub-colonies are replaced with new population and search process is continued.

4 Numerical Experiments

In this section, we show experimental results which validated our parallelized FF algorithm.

For testing purposes, we used following well-known unconstrained benchmark functions:

- Ackley
- Griewank
- Rastrigin
- Sphere

Because of the fact that these are standard benchmark functions, their definition is omitted.

All of the parallelization approaches have been implemented using Java programming language. In the Java programming language, concurrent programming is mostly concerned with threads. Threads are sometimes called lightweight processes. Both processes and threads provide an execution environment, but creating a new thread requires fewer resources than creating a new process. For test purposes, we created test application in Java programming language based on Lukasik and Zak software in Matlab. Tests were done on Intel I7-2600k processor with 8GB of RAM on Windows 7 Ultimate Edition x64 Operating System and NetBeans 7.0.1 Integrated Development Environment (IDE). In order to make the comparison clearer, values below E-12 were assumed to be 0. The parameters of algorithm are given in Table 1.

Parameter	Value
Number of function evaluation calls	1000000
Colony size	40
β_0	1
α	0.01
Υ	1
Number of runs with different seeds	30
Number of function parameters	5, 10, 50, 500

Table 1: Parameter settings

In the first series of our tests speed comparison is performed. It was our purpose to illustrate speed gains when every run of algorithm is separate thread. Comparison is done for different number of parameters of objective function. The results of speed test are shown in Table 2.

It is shown that speed gains are substantial for almost all combinations of number of parameters and benchmark functions. As the number of parameters increase, speed gains also increase.

The largest speed gains are obtained for 500 parameters. When test are done with small number of parameters, algorithm uses more CPU cycles for creating and synchronizing threads than for actual computation of objective functions. That explains small increase or even decrease in some cases, when 5 or 10 parameters are used. Since the *Sphere* function is a simple function, it requires small amount of CPU time when serial runs are used. More CPU time is used for creating and maintaining threads then for calculating *Sphere* function. The *Sphere* function is not so eligible for parallelization when the number of parameters is smaller than 500. Since this method of parallelization has no impact on quality of results, comparison of results are omitted.

Function	Number of parameters	Serial runs (seconds)	Parallel runs (seconds)	Speed increase
Ackley	5	**21.1**	22.0	0.957447
	10	30.4	**19.1**	**1.588235**
	50	121.3	**21.3**	**5.704225**
	500	1167.5	**224.3**	**5.204082**
Griewank	5	98.7	**80.2**	**1.230769**
	10	165.3	**100.8**	**1.639344**
	50	773.1	**121.8**	**6.346154**
	500	6913.9	**1143.1**	**6.048387**
Rastrigin	5	96.9	**69.6**	**1.391753**
	10	164.3	**105.2**	**1.5625**
	50	711.2	**132.2**	**5.37931**
	500	6873.5	**1200.4**	**5.726141**
Sphere	5	**39.5**	269.9	0.146341
	10	**69.0**	170.2	0.405405
	50	**629.8**	1053.9	0.59761
	500	6084.4	**4502.5**	**1.351351**

Table 2: Speed test results

Second part of our experiments aims for better results. We compared the results obtained when standard implementation of FF algorithm is run with multiple colony approach of the FF algorithm. These results are shown in Table 3.

Function		**NP = 5**		**NP = 10**		**NP = 50**	
		Standard	MC	Standard	MC	Standard	MC
Ackley	Best	8.2387E-9	0	4.4215E-7	2.4354E-9	3.4563E-4	8.1938E-6
	Mean	1.1714E-8	0	7.2289E-7	6.5691E-9	7.0238E-4	2.7645E-6
	Worst	2.0125E-8	0	3.7141E-6	4.2323E-7	5.7440E-3	4.1034E-5
	Stdev.	2.9338E-10	0	6.7591E-8	5.4301E-10	3.1484E-5	3.8367E-7
Griewank	Best	1.1481E-7	2.5786E-10	4.0202E-6	2.5454E-8	1.5263E-2	2.6657E-4
	Mean	5.7307E-7	5.0548E-9	4.2994E-6	8.1149E-8	7.4780E-2	9.2438E-4
	Worst	4.0935E-6	1.1812E-8	4.3398E-6	4.8359E-7	3.3232E-1	3.5610E-3
	Stdev.	1.4143E-8	3.3066E-9	3.1103E-10	3.1484E-11	6.5210E-3	5.2509E-5
Rastrigin	Best	3.4877E-7	2.0572E-9	3.2354E-6	5.1643E-8	4.6401	3.1751E-2
	Mean	4.1457E-7	4.6498E-9	2.6799E-5	2.1728E-7	5.7742	4.3045E-2
	Worst	1.2723E-6	3.1846E-8	5.5293E-5	3.5197E-7	7.8927	2.1609E-2
	Stdev.	4.2175E-8	2.3252E-10	8.3706E-7	4.1392E-9	6.8899E-1	4.2373E-4
Sphere	Best	0	0	8.0673E-12	0	7.7761E-10	2.0149E-11
	Mean	0	0	3.6363E-11	0	5.3148E-9	3.3036E-11
	Worst	0	0	1.9867E-10	0	2.3559E-9	7.5783E-11
	Stdev.	0	0	0	0	9.2001E-11	8.2987E-12

Table 3: Quality of results test

As we can see from Table 3, results obtained by multi-colony approach are always better than results generated by standard FF Algorithm.

Ratio between exploitation and exploration is more balanced in parallel implementation of FF algorithm than in its standard implementation.

4 Conclusion

In this paper parallelized the FF algorithm for unconstrained continuous optimization problems is introduced. The performance of this algorithm was measured through four tests on standard benchmark functions. We examined speed improvement, as well the quality of results improvement.

As we can from the comparative analysis between single threaded and multiple threaded FF algorithms, the multiple threaded one substantially outscored the traditional one. Parallelized FF algorithm obtained much better results in much less execution time. Parallelization overcomes deficiencies of single threaded execution.

As a plan for further research, we will try to enhance the original FF algorithm and parallelize it in the same way we did with the original one. Also, we will test our algorithm on more standard and less standard unconstrained functions, as well on constrained ones.

References:

[1] Fletcher R., *Practical Methods of optimization 2nd Edition*, Wiley, 2001, p.430.

[2] Cohen H., *Numerical approximation methods*, Springer, 2011, p. 485.

[3] Blum C., Roli A., *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*, ACM Comput. Surv., Vol. 35, Issue 3, 2003, pp. 268-308.

[4] Kennedy J., Eberhart R., *Particle Swarm Optimization*, Proceedings of IEEE International Conference on Neural Networks, 1995, pp. 1942–1948.

[5] Dorigo M, Maniezzo, *Ant Colony system: Optimization by a colony of cooperating agents*, IEEE Transactions on Systems, Man and Cybernetics - Part B, Vol. 26, Issue 1, 1996, pp. 29-41

[6] Kwee L., Yew-Soon O., Meng L., Xianshun C., Agarwal A., *Hybrid ant colony algorithms for path planning in sparse graphs*, Soft Computing, Vol. 12, Issue 10, 2008, pp. 981-994.

[7] D. Karaboga, *An idea based on honey bee swarm for numerical optimization*, Technical Report TR06, Computer Engineering, Department, Erciyes University, Turkey, 2005.

[8] Jiann-Horng L., Li-Ren H., *Chaotic bee swarm optimization algorithm for path planning of mobile robots*, Proceedings of the 10th WSEAS international conference on evolutionary computing, 2009, pp. 84-89.

[9] Yang, X. S. and Deb, S., *Cuckoo search via Lévy flights*, in: Proc. of World Congress on Nature & Biologically Inspired Computing (NaBIC 2009), 2009, pp. 210-214.

[10] Lukasik S., Zak S., *Firefly Algorithm for Continuous Constrained Optimization Tasks*, Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent, LNCS, Vol. 5796, 2009, pp. 97–106.

[11] Krishnanand K., Ghose D., *Glowworm swarm based optimization algorithm for multimodal optimization functions with collective robotics applications*, Multiagent and Grid Systems, Vol. 2, Issue 3, 2006, pp.209–222.

[12] Cecilia R., Tenreiro Machado J. A., *Crossing Genetic and Swarm Intelligence Algorithms to Generate Logic Circuits*, WSEAS Transactions on computers, Vol. 8, Issue 9, 2009, pp. 1419-1428.

[13] Zhuang X., Mastorakis N.E., *Edge Detection Based on the Collective Intelligence of Artificial Swarms*, in: Proceedings of the 4th WSEAS International Conference on Electronic, Signal Processing and Control, 2005, pp. 25-27.

[14] Stanger H. K. F., Lloyd J. E., Hillis D. M., *Phylogeny of North American fireflies (Coleoptera: Lampyridae): Implications for the evolution of light signals*, Molecular Phylogenetic sand Evolution, Vol. 45, Issue 1, 2007, pp. 33–49.

[15] Zang H., Zhang S., Hapeshi K., *A Review of Nature-Inspired Algorithms*, Jour. of Bionic Engineering, Vol. 7, Issue 3, 2010, pp. 232–237.

[16] X.S. Yang, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, 2008, p. 116.

[17] Pedemonte M., Nesmachnow S., Cancela H., *A survey of parallel ant colony optimization*, Applied Soft Computing, Vol 11., Issue 8, 2011, pp. 5181-5197.