A Multithreaded Implementation of the Fish School Search Algorithm

Conference Paper *in* Communications in Computer and Information Science · May 2014

DOI: 10.1007/978-3-319-12745-3_8

2 authors:

Marcelo Gomes Pereira de Lacerda
Entos
24 PUBLICATIONS **115** CITATIONS
SEE PROFILE

Fernando Buarque de Lima Neto
University of Pernambuco
235 PUBLICATIONS **1,886** CITATIONS
SEE PROFILE

A Multithreaded Implementation of the Fish School Search Algorithm

Marcelo Gomes Pereira de Lacerda[1], Fernando Buarque de Lima Neto[1,2]

[1,] University of Pernambuco, Recife, Pernambuco, Brazil
{mgpl,fbln}@ecomp.poli.br

[2] Westfälische Wilhelms-Universität Münster, Chair for Information Systems and Supply Chain Management, 48149 Münster, Germany
buarque@uni-muenster.de

Abstract. This work introduces a multithreaded implementation of the Fish School Search (FSS) algorithm, the Multithreaded Fish School Search (MTFSS). In this new approach, each fish has its behaviour executed within an individual thread, of which creation, execution and death are managed by the runtime environment and the operating system. Five well-known benchmark functions were used in order to evaluate the speed-up of the MTFSS in comparison with the standard FSS and check if there are statistically significant changes in the ability of the new algorithm to find good solutions. The experiments were carried out in a regular personal computer as opposed to expensive set ups and the results showed that the new version of the algorithm is able to achieve interesting growing speed-ups for increasingly higher problem dimensionalities when compared to the standard FSS. This, without losing the ability of the original algorithm of finding good solutions and without any need of more powerful hardware (e.g. parallel computers).

Keywords: Computational Intelligence; Swarm Intelligence; Fish School Search; Multithreads.

1 Introduction

Optimization tasks are present in many situations where information technology is required. Managers, for example, must take decisions aiming to maximize the company's profit. Racing teams adjust their cars in a way that they will have the best performance given the limits of the machine. These are just some real life examples where optimization tasks are required. Formally, optimization should be understood as a search in which system adjustments according to the utility function are carried out aiming to obtain the best possible outcome (definition extended from Engelbrecht [1]).

Optimization algorithms are computational techniques that search for solutions of problems represented by an objective function. Up to the end of the 1980s, exact optimization algorithms were considered the definitive methods for solving optimization problems. Although significant advances were made on these approaches, it was found

that for highly dimensional real-world problems (*e.g.* Supply Chain Network Planning problem [2]), these methods may take an exponentially increasing large amount of time to be solved. Thus, some instances can become intractable.

As an alternative, approximate approaches have been developed in order to face this issue. These techniques do not guarantee the best output possible, but most of time, good enough ones are normally the case.

In this context, nature inspired techniques have been developed. A successful set of these techniques are known as population based algorithms (PBA), due to their characteristics of using a group of artificial entities to collectively and in a coordinated way perform the search.

Swarm Intelligence (SI) can be referred to as the property of any system in which the interaction between very simple components generates complex functional patterns [4]. Within the field of Computational Intelligence, many PBAs present this most interesting behavior. These algorithms form the Swarm Intelligence subfield. Some of the best known algorithms within SI are: Particle Swarm Optimization (PSO) [5], Ant Colony Optimization (ACO) [6], Artificial Bee Colony (ABC) [7], Bacterial Foraging Algorithm (BFA) [8] and Fish School Search (FSS) [9].

The quality of the best solutions found by these algorithms depends on the number of analyzed solutions and, consequently, on the execution time, which means that, in order to acquire better solutions, a longer running time is necessary [3].

Even for some metaheuristics, by increasing the number of dimensions of an optimization problem, one observes an increase of its search space's complexity. Therefore, a longer running time will be needed for satisfactorily solving these problems. This causes a significant growth in the execution time of the whole optimization process.

Efforts have been made on the creation of parallel versions of SI algorithms. Two main approaches were found in the literature: GPU based parallel algorithms (multicore parallel algorithms), and multithread parallel algorithms. Most of the GPU based parallel algorithms achieved outstanding speed-ups due to the extremely high number of cores available (*e.g.* [10][11]).

Within the multithread parallel algorithms, two approaches were found: cluster based parallel algorithms and single machine parallel algorithms. In comparison to the GPU based approaches, the cluster based parallel algorithms achieved intermediate results, since there are usually a smaller number of available cores when compared to approaches based on GPUs, besides there is a significant adding cost due to the need of communication between processors (*e.g.* [3]). Obviously, the single machine parallel algorithms have been achieved the worst results, in terms of speed-up, among all approaches (*e.g.* [12][13]). However, the main advantage of this approach is that it does not require the acquisition of a complex to run or more expensive system in order to achieve speed-ups. Moreover, the later produces easier coding algorithms than the GPU and cluster based approaches.

Fish School Search (FSS), which was proposed by Bastos Filho and Lima Neto in 2008, is, in its basic version, an unimodal optimization algorithm inspired on the collective behavior of fish schools. The mechanisms of feeding and coordinated movement were used as inspiration to create the collective search mechanism. The main idea is to make the fishes (candidate solutions) to swim toward the direction of the positive gradient in order to gain weight. Collectively, the heavier fishes are more influent in the search process as a whole as the barycenter of the school gradually moves

towards better places in the search space. It was firstly designed to run in a single thread.

The first parallel version of the FSS algorithm was proposed by Lins in 2012 [10]. In this work, a GPU based approach (*i.e.* multicore approach) was used in order to speed-up the execution of the FSS algorithm. In the experiments, aiming to evaluate the proposed approach in different architectures, two different machines were used: MacBook Pro with one GPU Nvidia GeForce 320M with 48 cores and one Personal Super Computer (PSC) with 4 GPUs Tesla C2070 with 448 cores each one, all working in parallel. All GPUs used in the experiments are compatible with CUDA architecture. The first one was equipped with an Intel processor with 2 cores and the latter with 4 Intel processors with 4 cores each one. There the authors achieved a speed-up up to 127.9006 with all GPUs working together, besides its performance in terms of fitness of the best solution found was significantly improved in comparison with the FSS algorithm.

This work proposes a single machine multithread version of the Fish School Search algorithm. In this version, there is no need for a more expensive and more complex platform in order to run the referred algorithm. The same machine that was only able to run the standard version of the FSS was able to run the new faster approach, presented here. This new version was specially designed for highly dimensional problems, usually the real-world ones, due to the large execution time needed to optimize such problems.

This paper is organized as follows: on Section II, the standard Fish School Search algorithm is explained; on Section III, the Multithreaded Fish School Search is presented; on Section IV, the experiments performed in this work are described; on Section V, the results are showed and discussed; and finally, on Section VI, a conclusion about this work is made and all future work is presented.

2 Fish School Search

Fish School Search is inspired by the collective behaviour of natural fish schools. In fish schools, the individuals work collectively as a single organism but do possess some local freedom. This combination accounts for fine as well as greater granularities during their search for food.

In FSS, the success of the search process is represented by the weight of each fish. In other words, the heavier is an individual, the better is its represented solution. The weight of the fish is updated throughout the feeding process. A second means to encode success in FSS is the radius of the school. It is noteworthy to mention that by contracting or expanding the radius of the school, FSS can automatically switches between exploitation and exploration, respectively. The pseudo-code of FSS is provided by Algorithm 1.

2.1 Individual Movement Operator

In the Individual Movement Operator, each fish moves randomly and independently, but always toward the positive gradient. In other words, the fish moves only if the new position is better than the previous one, with regards to the objective function.

This movement is described by (1), where $x_{ij}(t+1)$ is the new value of the dimension j in the position vector of the individual i, $x_{ij}(t)$ is the old value, $rand(0,1)$ is a random value between 0 and 1 and $step_{ind}(t)$ is the step size on time t. The new step size is calculated through (2), where $step_{ind_{init}}$ and $step_{ind_{final}}$ are the initial and final step sizes and $iterations$ is the maximum number of iterations.

$$\vec{x}_i(t+1) = \vec{x}_i(t) + rand(0,1)\, step_{ind}(t), \tag{1}$$

$$step_{ind}(t+1) = step_{ind}(t) - \frac{step_{ind_{init}} - step_{ind_{final}}}{iterations}. \tag{2}$$

```
P: Fish population;
while Stopping condition is not met do:
  for each fish in P do:
    Run the individual movement;
  for each fish in P do:
    Run the feeding process;
  for each fish in P do:
    Run the collective instinctive movement;
  Calculate the fish school's barycenter;
  for each fish in P do:
    Run the collective volitive movement;
Return the best solution found;
```

Algorithm 1**.** Pseudo-code of FSS.

2.2 Feeding Operator

As mentioned before, the feeding operator is responsible for the weight update of all fishes. This update process is defined by (3), where Δf_i, is the fitness variation after the Individual Movement of the fish i, and $\max(\Delta f)$ is the maximum fitness variation in the whole population.

$$W_i(t+1) = W_i(t) + \frac{\Delta f_i}{\max(\Delta f)}. \tag{3}$$

2.3 Collective Instinctive Movement Operator

The Collective Instinctive Movement Operator is the first collective movement in the algorithm. Every fish performs this movement by adding a vector to its current position, which is calculated according to (4), where N is the population size, $\Delta\vec{x}_k$ and $\Delta f(\vec{x}_k)$ are the position variation and the fitness variation of the individual of index in the Individual Movement, being this vector common to all fishes. The final movement is defined by (5).

$$\vec{I} = \left(\frac{\sum_{k=1}^{N} \Delta\vec{x}_k \Delta f(\vec{x}_k)}{\sum_{k=1}^{N} \Delta f(\vec{x}_k)}\right), \tag{4}$$

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{I}. \quad (5)$$

2.4 Collective Volitive Movement Operator

In this step, the population must contract or expand, using as reference the bary-center of the fish school, which is calculated according to (6), where $W_i(t)$ is the weight of the fish i on time t. The total weight of the whole population must be calculated in order to decide if the fish school will contract or expand. If the total weight increased after the last Individual Movement, the school as a whole will contract in order to execute a finer search, which means that the search process has been successful. Otherwise, the population will expand, meaning that the search process is not qualitatively improving. This could be due to a bad region of the search space or the school is trapped in a local minimum (hence, it should try to escape from it). The contraction and expansion processes are defined by (7) and (8), respectively, where, $distance\left(\vec{x}_i(t), \vec{B}(t)\right)$ is the Euclidian distance between the vector $\vec{x}_i(t)$ and $\vec{B}(t)$ and $step_{vol}$ is the volitive step size, which must be defined by the user.

$$\vec{B}(t) = \frac{\sum_{i=1}^{N} \vec{x}_i W_i(t)}{\sum_{i=1}^{N} W_i(t)}, \quad (6)$$

$$\vec{x}_i(t+1) = \vec{x}_i(t) - step_{vol} rand(0,1) \frac{\left(\vec{x}_i(t) - \vec{B}(t)\right)}{distance\left(\vec{x}_i(t), \vec{B}(t)\right)}, \quad (7)$$

$$\vec{x}_i(t+1) = \vec{x}_i(t) + step_{vol} rand(0,1) \frac{\left(\vec{x}_i(t) - \vec{B}(t)\right)}{distance\left(\vec{x}_i(t), \vec{B}(t)\right)}. \quad (8)$$

3 Multithread Fish School Search

The Multithreaded Fish School Search (MTFSS) algorithm is designed to run a single machine, splitting the search process into parallel threads in order to reduce the running time of the optimization task. In this algorithm, the behaviour of each fish in the population is executed in one individual thread. The pseudo-code of the algorithm performed by each fish is showed in Algorithm. 2.

The individual movement is performed exactly like it is made in the standard FSS algorithm. After this operator, the fish calls a sub-routine called *Barrier1*. This sub-routine is showed in Algorithm. 3.

It is important to mention that algorithm was coded in Java programming language version 7. Therefore, the considered *static* modifier used in the pseudo-code of *Barrier1* sub-routine makes the instance of the variable *barrier1_counter* unique for all fishes. Moreover, the **wait()** and **notifyAll()** functions make the fish that calls the sub-routine *Barrier1* enter in WAITING state and wake up all the other fishes, putting them in RUNNABLE state again, respectively. It means that every fish that reaches this sub-routine must wait for the other ones. The last fish to reach this barrier finds the maximum fitness variation, shares this information with the others, calculates the vector $\vec{I}$ and wakes-up the rest of the population.

```
while Stopping condition is not met do:
  Perform the individual movement;
  Barrier1();
  Perform the feeding process;
  Barrier2();
  Perform the collective instinctive movement;
  Perform the collective volitive movement;
```

Algorithm 2. Algorithm performed by each fish (*i.e.* each thread).

```
static barrier1_counter:Number of fishes that is waiting
on Barrier1();
population_size:Number of fishes in the population;
Barrier1():
  If barrier1_counter<population_size:
    barrier1_counter = barrier1_counter+1;
    wait();
  Else:
    Find maximum fitness variation and store in a global
    variable, which is accessible by all fishes;
    Calculate I vector for collective instinctive move-
    ment;
    notifyAll();
```

Algorithm 3. *Barrier1* sub-routine.

After all fishes are in RUNNABLE state again, each one calculates its new weight throughout the feeding operator. After the feeding operator, the *Barrier2* sub-routine is called. This sub-routine is showed in Algorithm. 4.

After all fishes are in RUNNABLE state again, the collective instinctive and volitive movements are performed exactly like the standard FSS algorithm, using the vector $\vec{I}$, the barycentre and fish school total weight calculated by the last fish reaching *Barrier2* sub-routine. However, there is a small change in the barycentre calculation, but still important: all fishes' positions used for this purpose are the ones right after the execution of the Individual Movement, instead of being the positions after the Insitinctive Movement.

These barriers are intended to keep a minimum synchronism among the fishes. The only change in the algorithm's rationale in comparison to the standard version is the use of the positions after the individual movement in the calculation of the barycentre, instead of using the positions after the collective instinctive movement. Results, which are presented in Section V, proved that this change did not produce significant changes in the algorithm's ability to find good solutions.

It is important to mention that all threads' execution, creation and death are automatically managed by the runtime environment and the operating system, which in itself is another facilitator for the adoption of the multi-threaded version of the FSS algorithm.

```
static barrier2_counter:Number of fishes that is waiting
on Barrier2();
population_size:Number of fishes in the population;
  Barrier2():
    If barrier2_counter<population_size:
      Barrier2_counter = barrier2_counter+1;
      wait();
    Else:
      Calculate weight sum of all fishes;
      Calculate barycentre, taking into account the posi-
      tion of each fish after the individual movement;
      notifyAll();
```

Algorithm 4. *Barrier2* sub-routine.

4 Experiments Description

This work aims to compare the running time and the ability of finding good solutions of the MTFSS with the FSS algorithm. Five well known benchmark functions were used for this purpose: Rastrigin, Rosenbrock, Griewank, Ackley and Schwefel 1.2, which are described in [9]. The boundaries of the search space of these functions were set to [-5.12,5.12], [-30,30], [-600,600], [-32,32], [-100,100], respectively. The initialization subspaces for each function were set to [2.56,5.12], [15,30], [300,600], [16,32], [50,100]. The initialization subspace defines the region in the search space in which all individuals must be initialized.

Thirty executions were performed for each experiment. For both algorithms, the individual step size and the W_{scale} factor were set to 10% of the search space length, at the beginning of the optimization task, linearly decreasing to 0.001%, at the end of the process, and 5000, respectively. The volitive step size was set to twice the individual step size. For the evaluation of the best fitness found, 30 fishes, 30 dimensions for each function and 5000 iterations were used. For running time evaluation, the number of dimensions, individuals and iterations were varied, as shown on Section V, but only Rastrigin function was used. The execution time were measured in milliseconds for each execution. The speed-up values are the average value of the speed-ups in the thirty executions for each combination between dimensions number and population size.

All experiments were performed in a personal computer equipped with a processor Intel Core i5 650 3.2 GHz (2 physical and 2 virtual cores), 4GB DDR3 RAM memory. The operational system installed was Ubuntu 13.04 32-bits, with Java version 1.7.0_25 and OpenJDK 7 Runtime Environment IcedTea 2.3.10.

5 Results and Discussion

In this section, all results acquired throughout the experiments are presented.

Figs. 1, 2 and 3 show the speed-ups achieved by the MTFSS in comparison to the running time of the FSS algorithm for 100, 1000 and 10000 iterations, respectively. It is possible to see that for 10 and 100 dimensions, the FSS is faster than the MTFSS. However, throughout the approximation of the linear functions that are formed by the points that represents the results of the experiments for 100 and 1000 dimensions, it can be also perceived that the MTFSS is quite faster than the FSS when the problem has approximately more than 203, 309 and 297 dimensions for 100, 1000 and 10000 iterations, respectively. Since there are no results for experiments with the number of dimensions between 100 and 1000, linear functions were used in order to acquire these approximations. The linear functions used for this purpose were: $f(x) = 0.00127(x - 100) + 0.871$, $f(x) = 0.00167(x - 100) + 0.65$ and $f(x) = 0.00165(x - 100) + 0.6746$.

Since the MTFSS was designed to reduce the running time of the FSS algorithm in cases where the standard version is taking too long to complete the optimization task, which are the case of highly dimensional problems, it is possible to say that the MTFSS performs satisfactorily those tasks without any technical or economical burden. It is clear also that the best results were accquired in the experiments with larger dimensions, e.g. over 1000 dimensions; continuing to improve even further for 10,000 dimensions.

It was not observed significant differences between the experiments with 1000 and 10000 iterations. Even though the experiments with 100 iterations presented some significantly different results from the ones for 10, 100 and 1000 dimensions, for 10000 dimensions the results were quite similar. The three highest speed-ups were achieved in the experiments with 100 individuals and 10000 dimensions for 100, 1000 and 10000 iterations: 2.7695, 2.7668 and 2.7234, respectively.

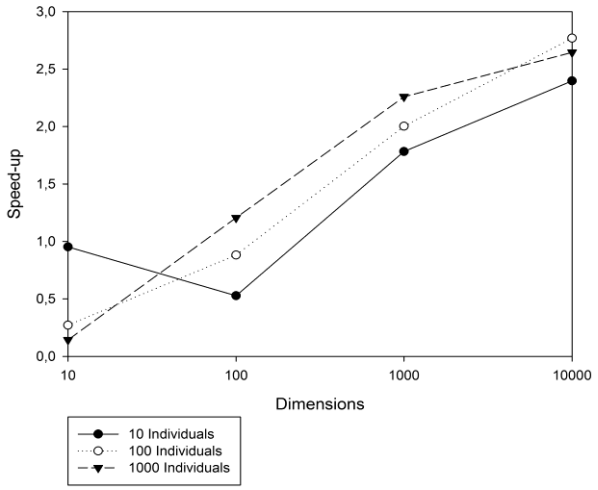

Fig. 1. Speed-up for 100 iterations.

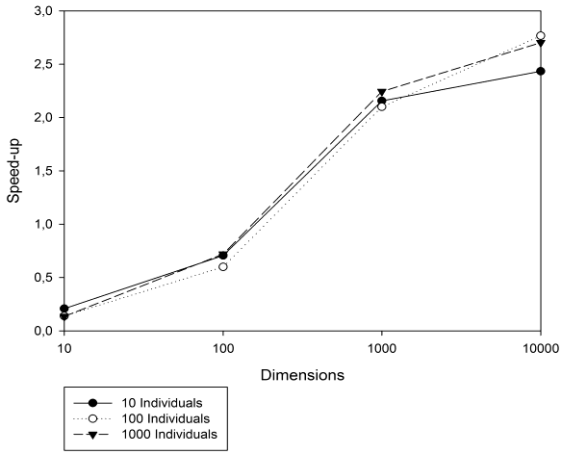

Fig. 2. Speed-up for 1000 iterations.

Figs. 4, 5, 6, 7 and 8 shows comparisons between the FSS and MTFSS algorithms in terms of best fitness found. The box-plot graphs represents 30 executions with the setup presented on Section IV in each function. Observing these graphs in conjunction with the Wilcoxon test results that are presented on Table I, it can be affirmed that there is no significant difference in terms of best fitness found between the algorithms FSS and MTFSS. Therefore, it can be concluded that the MTFSS is able to speed-up the FSS execution up to approximately 3 times, without losing the ability of finding good solutions of the original algorithm.

On Table 1, ▲, ▼ and ▬ mean that the MTFSS performed statistically better, worse or equal to the FSS, respectively.

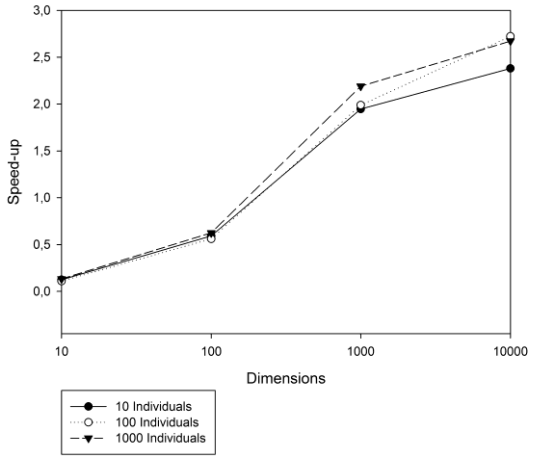

Fig. 3. Speed-up for 10000 iterations.

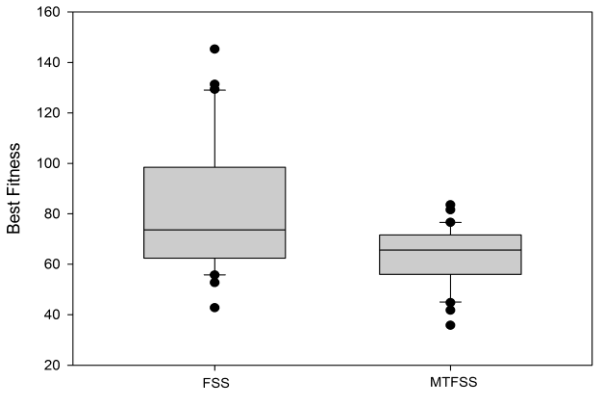

Fig. 4. Best fitness found comparison for Rastrigin

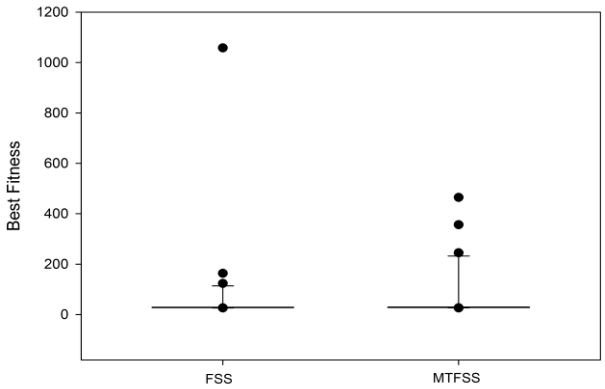

Fig. 5. Best fitness found comparison for Rosenbrock.

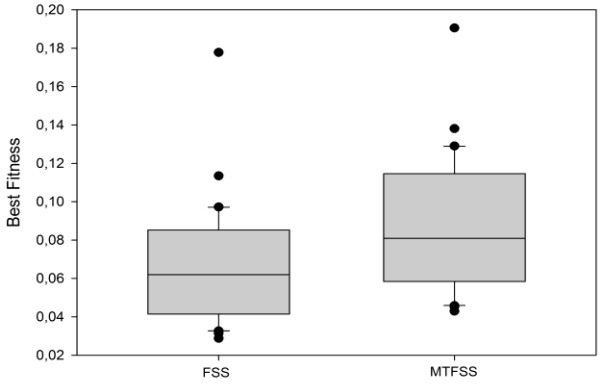

Fig. 6. Best fitness found comparison for Griewank.

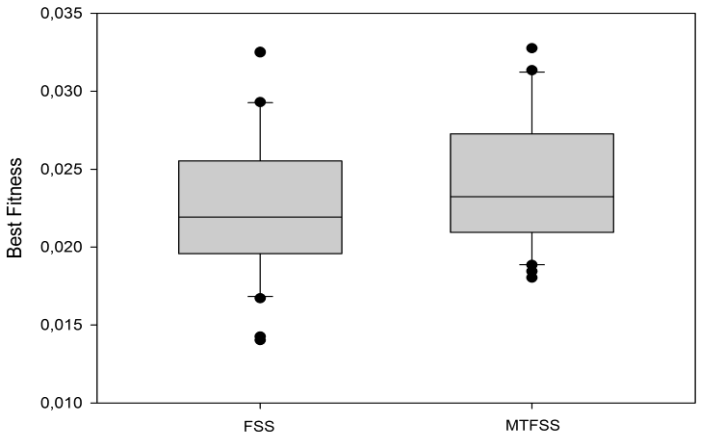

Fig. 7. Best fitness found comparison for Ackley.

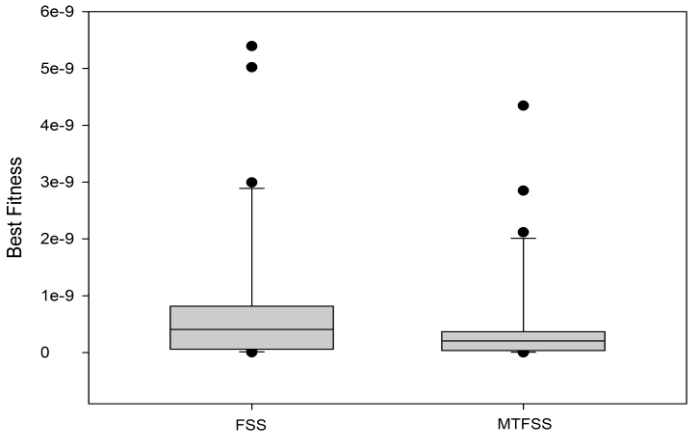

Fig. 8. Best fitness found comparison for Schwefel 1.2.

Table 1. Wilcoxon test results for best fitness found, comparison between MTFSS and FSS. ($p = 0.05$)

Objective Function	***FSS***
Rastrigin	▲
Rosenbrock	—
Griewank	▼
Ackley	—
Schwefel 1.2	—

6 Conclusion and Future Work

In this paper, it was presented a multithreaded version of the FSS algorithm, the Multithreaded Fish School Search algorithm (MTFSS). Differently from the pFSS algorithm, which is the first parallel version of the FSS (GPU based implementation), the parallelism of the MTFSS is based on threads, instead of cores, where the behaviour of each fish is run by a single thread. So that, the new approach was designed to run in any personal computer with multithread processing available, without the need for GPUs or even, of a cluster.

The results showed that the algorithm is able to achieve interesting speed-ups in comparison to the standard FSS, without losing the ability of the original algorithm of finding good solutions.

As future work, the authors of this work intend to test the algorithm in computers with more powerful processors; test the algorithm in different operating systems; and test the algorithm with different programming languages.

References

1. A. P. Engelbrecht. Computational Intelligence, An Introduct., 1st ed., Wiley & Sons, 2007.
2. L. F. A. Pessoa, D. Horstkemper, D. S. Braga, B. Hellingrath, M. G. P. Lacerda, F. B. Lima Neto, "Comparison of Optimization Techniques for Complex Supply Chain Network Planning Problems". In: Anais do Congresso Nacional de Pesquisa e Ensino em Transporte (ANPET), Belém-Brazil, 2013.
3. W. Bozejko, J. Pempera, and C. Smutnicki, "Multi-thread parallel metaheuristics for the flow shop prblem", Artificial Intelligence and Soft Computing, 2008, pp.454-462.
4. E. Bonabeau, M. Dorigo, and G. Theraulaz, Swarm Intelligence: from natural to artificial systems. Oxford University Press, Inc., 1999.
5. J. Kennedy and R. Eberhart, "A new optimizer using particle swarm theory", Micro Machine and Human Science, International Symposium on, 1995, pp. 39-43.
6. M. Dorigo. Optimization, learning and natural algorithms. PhD Thesis – Politecnico di Milano, 1992.
7. D. Karaboga & B. Basturk "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm", Global Optimization, 2006, pp. 459-471.
8. K. M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control.", IEEE Control Systems Magazine, v. 22, n. 3, 2002, pp. 52–67.
9. C. J. A. B Filho, F. B. de Lima Neto, A. J. C. C.. Lins, A. I. S. Nascimento., and M. P. Lima, , "A novel search algorithm based on fish school behavior," Systems, Man and Cybernetics, SMC 2008. IEEE International Conference on, 2008, pp. 2646-2651.
10. A. J. C. C. Lins, Paralelização de Algoritmos de Otimização baseados em Cardumes através de Unidades de Procesamento Gráfico, MSc. Thesis, University of Pernambuco, 2012.
11. K. Ding, S. Zheng, Y. Tan, "A GPU-based parallel fireworks algorithm for optimization". In: Genetic and Evolutionary Computation Conference, GECCO 2013, 2013, pp. 9-16.
12. N. Bacanin, M. Tuba, I. Brajevic, "Performance of object-oriented software system for improved artificial bee colony optimization", International Journal of Mathematics and Computers in Simulation, 2011, pp. 154-162.
13. M. Tuba, N. Bacanin, N. Stanarevic, "Multithreaded implementation and performance of a modified artificial fish swarm algorithm for unconstrained optimization". International Journal of Mathematics and Computers in Simulation, 2013, pp. 215-222.