Prog Artif Intell (2014) 2:193–203
DOI 10.1007/s13748-014-0052-7

REGULAR PAPER

Fisherman search procedure

Oscar J. Alejo-Machado · Juan M. Fernández-Luna · Juan F. Huete · Eduardo R. Concepción Morales

Received: 10 December 2012 / Accepted: 10 January 2014 / Published online: 12 April 2014

Abstract Optimization is used in diverse areas of science, technology, and business. Metaheuristics are one of the common approaches for solving optimization problems. In this paper, we propose a novel and functional metaheuristic Fisherman Search Procedure (FSP), to solve global optimization problems, which explores new solutions using a combination of guided and local search. We evaluate the performance of FSP on a set of benchmark functions commonly used for testing global optimization methods. We compare FSP with other heuristic methods referenced in the literature, namely differential evolution (DE), particle swarm optimization (PSO), and greedy randomized adaptive search procedures. Results are analyzed in terms of successful runs, i.e., convergence on global minimum values, and time consumption, demonstrating that FSP can achieve very good performances in most of the cases. In 90 % of the cases, FSP is located among the two better results as for successful runs. On the other hand, with regard to time consumption, FSP shows similar results to PSO and DE, achieving the best and second best results for 82 % of the test functions. Finally, FSP showed to be a simple and robust metaheuristic that achieves good solutions for all evaluated theoretical problems.

Keywords Metaheuristics · Global optimization · Search procedures

O. J. Alejo-Machado (✉) · E. R. Concepción Morales
Departamento de Informática, Universidad de Cienfuegos, Cienfuegos, Cuba
e-mail: alejo@ucf.edu.cu

J. M. Fernández-Luna · J. F. Huete
Departamento de Ciencias de la Computación e I.A., E.T.S.I. Informática y de Telecomunicación, CITIC-UGR, Universidad de Granada, Granada, Spain

1 Introduction

Metaheuristic techniques are used to solve hard problems (generally NP-hard problems) instead of, or in conjunction with, exact algorithms. When the problem dimension becomes very large, exact algorithms may not be useful since they are computationally too expensive. In these cases, approximate algorithms (which do not guarantee to find an optimal solution) are a very effective alternative. Metaheuristics are approximate algorithms, which encompass and combine constructive methods, local search strategies, local optima escaping strategies, and population-based search. They include, but they are not restricted to: tabu search [1], simulated annealing [2], evolutionary computation [3], memetic algorithms [4], scatter search [5], iterative improvement (Hill climbing et similia), ant colony optimization [6], particle swarm optimization (PSO) [7], greedy randomized adaptive search procedure (GRASP) [8], iterated local search [9,10], and variable neighborhood search [11].

Metaheuristics are strategies to design heuristic procedures. Therefore, metaheuristics types are defined according to the type of procedures referred by them. Some of the fundamental types are metaheuristics for relaxation procedures, constructive processes, neighborhood searches, and evolutive procedures.

Some metaheuristics arise by combining different types of metaheuristics, as GRASP [8] that combines a constructive phase with a phase of search improvement. Other metaheuristics are centered in the use of some type of computational resource or special formulation, such as neural networks, ant systems, or the constraint programming, and they are not included clearly in none of the four previous types.

In general, one way or another, all metaheuristics can be conceived as strategies applied to search processes. In fact, in this work, a new metaheuristic, named Fisherman

Search Procedure (FSP), is introduced. This search algorithm explores new solutions using a combination of guided search and local search. This metaheuristic was designed with the purpose of developing useful and practical solutions for a wide variety of global optimization problems.

The main advantages are the following: easy implementation; it provides an explicit description of the model-based idea and the own conception of this metaheuristic allows it to be applied to a large number of classical global optimization problems, as well as to those that arise in real-world situations in different areas of applied sciences, engineering, and economy.

In order to introduce our metaheuristic, as well as the experimentation to evaluate its performance, the rest of the paper is organized as follows. In Sect. 2, the FSP Approach is formally introduced. The parameterization phase is explained in Sect. 3. Section 4 is devoted to describe the settings of the experimental study that allows us to evaluate the performance and the computational time of the new proposed algorithm, as well as the results obtained, comparing them with several metaheuristics referenced in the literature. In Sect. 5, four FSP extensions are proposed for further discussion. Finally, in Sect. 6, we conclude our work and point out some directions for future research.

2 FSP approach

In this section, a new metaheuristic for global optimization problems is formally described. FSP is a global optimization method, inspired by the cognitive behavior and the dexterities observed in a fisherman. This method explores new solutions using a combination of guided search and local search. The algorithm is shown in Fig. 1.

Initially, we set N capture points in the whole fishing area (search space). Basically, each capture point is composed of a position vector, x_i (located in the search space), and a memory of the best solution found by the fisherman in the neighborhood of the capture point p_i. Let $x_i \in X$, where $X = \{x_1, x_2, \ldots, x_N\}$ denotes the set of position vectors of the capture points.

The fisherman global memory is defined as g_{best} (e.g., the best among the capture points). Following a fishing trajectory, the fisherman throws the fishing network L times in each capture point.

The fishing network is described by a set of position vectors $y_{ij} \in Y$, $Y = \{y_{i1}, y_{i2}, \ldots, y_{iM}\}$ created and defined starting from a reference point, where y_{ij} is the jth vector in the ith capture point, $i = 1, 2, \ldots, N$; $j = 1, 2, \ldots, M$. The value M is the quantity of network position vectors. Each position vector represents a knot of the traditional fishing network.

```
1:  Input: T, N, L and M
2:  for i=1 to N do
3:    Initialize x_i
4:    Evaluate x_i
5:    Instantiate p_i
6:    Update g_best
7:  end for
8:  for i=1 to T do
9:    for j=1 to N do
10:     while ((L > 0) and (find improvements)) do
11:       for k=1 to M do
12:         Update y_jk with expression y_jk = x_j + A_k
13:         Evaluate y_jk
14:         Update p_j
15:       end for
16:       L = L − 1
17:     end while
18:     Update x_j
19:     Update g_best
20:   end for
21:   Apply a strategy to update the width coefficient c
22: end for
23: Output: g_best
```

Fig. 1 The FSP algorithm

The equation used to create the network position vectors is the following:

$$y_{ij} = x_i + A_j, \tag{1}$$

where A_j is an N-dimensional vector composed of random numbers in the range $[-c, c]$, where c is a real number denominated width coefficient.

The network position vectors are evaluated, and if some of them has a fitness value better than the p_i value of the capture point from where the launching was performed, the capture point is updated with this new position, and the fisherman network is redefined again considering this better solution as the new reference vector. When updating the p_i value for the ith capture point, if p_i is better than g_{best}, the latter is also updated.

The whole fishing procedure is repeated T times. After each iteration, the improvement rate in each capture point is analyzed. If the function value in p_i for the ith capture point improves with respect to previous iteration, we recommend to decrease the value of the width coefficient, with the idea of reducing the risk of skipping the global minimum, in the case when the solution is close to it. This decrease of the width

coefficient should be made multiplying by a factor of 0.95 the c value, to guarantee a gradual approximation to global maximum.

This strategy pursues the intuitive idea of narrowing the holes of the fishing network to guarantee the capture. On the other hand, if the function value in p_i for the ith capture point does not improve, the value of the coefficient of width should be increased its initial value twice, with the idea of looking for an abrupt change that accelerates the exploration of new fishing areas (search spaces). In this last case, if a better function value is found, we update the p_i value for the ith capture point and reset the width coefficient to its initial value.

3 Parameterization

In the parameterization study, every parameter of the metaheuristic is evaluated and analyzed. With this goal in mind, we studied the behavior of the algorithm parameters considering De Jong's five functions test, which are widely used in these cases.

The configurations of this functions exhibit a balance between complexity and diversity; balance that is necessary to study the heuristic. The dimensions (n), the searching region (S), and the minimum (min) of each one of the functions can be observed in Table 1 and their graphics in Fig. 2.

Functions f_1 and f_2, known as the Sphere and Rosenbrock functions, respectively, are convex one-modal functions for two dimensions. Function f_3 is known as the step function; it is discontinuous, convex with a single minimum. Function f_4 corresponds to the function with Gaussian noise that is convex, one-modal noisy. Finally, function f_5, known as Shekel's foxholes, proposed by Shekel, is a continuous function, non convex, two-dimensional, no square [12].

Our algorithm uses five independent parameters: the number of iterations T, the capture points N, the throws L, the quantity of position vectors M that represent the fishing network, and the width coefficient c. The amount of iterations has a stopping criteria according to the convergency of the algorithm: x numbers of iterations without improving the result will terminate the process. For the tests, this criteria has been set to the 10 % of the given number of iterations.

3.1 Capture points

The number of the capture points is an important factor in the algorithm stability. We carried out tests of the internal heuristics considering a range from 1 to 100 capture points; the results are shown in Fig. 3. In the case of De Jong's five functions test suite analyzed, we observed that the average of the minimum found stabilizes for values between 80 and 100; our recommended range of values.

3.2 Number of iterations and throws

The number of throws and iterations controls the searching extension; a small number of iterations and throws means a quick search. It is better to use the smallest number for which the algorithm can stabilize and find an answer within the desired boundaries.

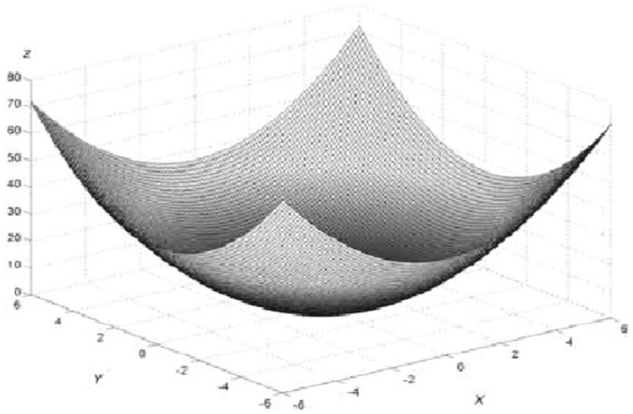
In Table 2, we can see the average value that the heuristic achieves on these test functions on 30 runs of the algorithm, for a series of guided combinations of the parameter that are analyzed in this subsection.

For functions f_1 and f_2, we observe that the increase in the number of iterations gradually increases the algorithm precision. In both cases, we can also see that with five throws, the algorithm exhibits a satisfactory stable performance.

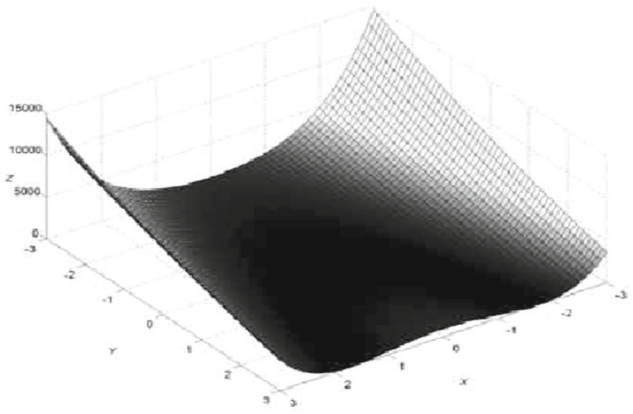
For function f_3, the algorithm always reaches the function minimum. For function f_4, the algorithm for different parameter combinations does not improve or worsen its initial precision with respect to the function minimum.

Table 1 Benchmark functions for the parameterization phase

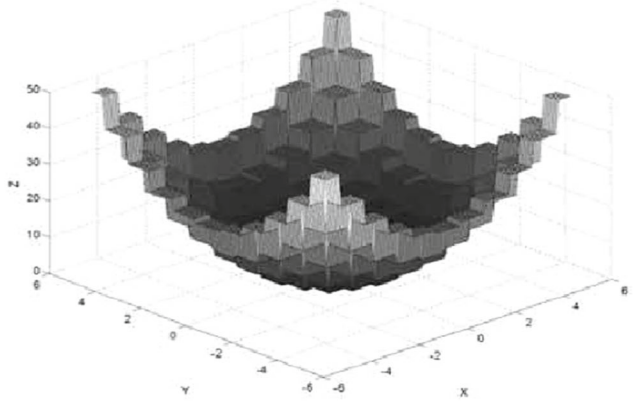
Functions	n	S	Min
$f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2$	2	$[-500, 500]^n$	0
$f_2\left(\vec{x}\right) = \sum_{i=1}^{n} 100\left(x_i^2 - x_{i+1}\right)^2 + \left(1 - x_i^2\right)^2$	2	$[-5.12, 5.12]^n$	0
$f_3\left(\vec{x}\right) = \sum_{i=1}^{n} \left(\lfloor x_i + 0.5 \rfloor\right)^2$	2	$[-5.12, 5.12]^n$	0
$f_4\left(\vec{x}\right) = \sum_{i=1}^{n} i\left(x_i^4\right) + \text{Gauss}\,(0, 1)$	2	$[-1.28, 1.28]^n$	0
$f_5\left(\vec{x}\right) = \frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}\left(x_i - a_{ij}\right)^6}$	2	$[-65.536, 65.536]^n$	≈ 1
$a_{ij} = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & \ldots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & -16 & \ldots & 32 & 32 & 32 \end{pmatrix}$			



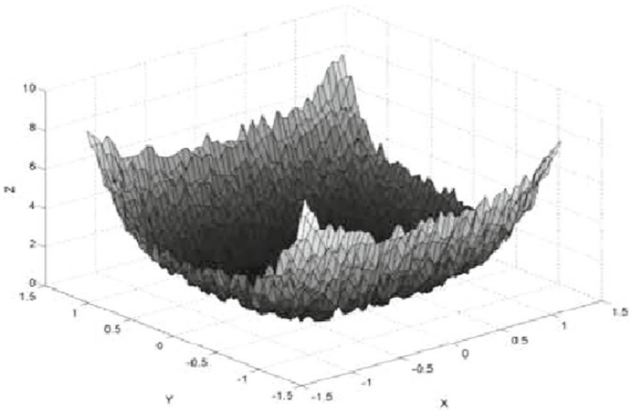
(a) f_1: Sphere Model



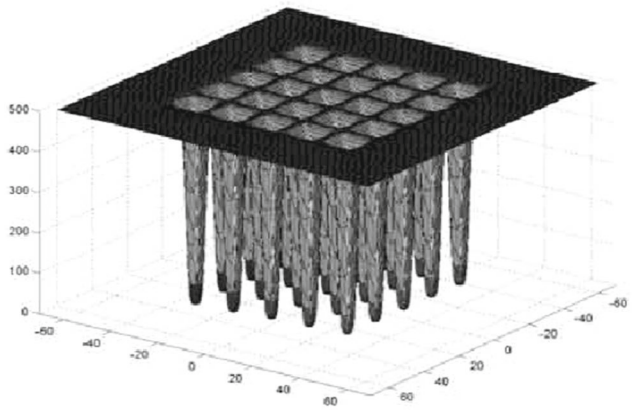
(b) f_2: Rosenbrock's Function



(c) f_3: Step Function



(d) f_4: Function with Gaussian Noise



(f) f_5: Shekel's Foxholes

Fig. 2 Test functions for the parameterization

For function f_5, the algorithm is not affected too much by the values of T and L and reaches the minimum in all tests.

This promising results are obtained in part because the number of capture points and the width coefficient values were set according to the best achieved range in the tests that were carried out in Subsects. 3.1 and 3.3.

3.3 Fishing network and the width coefficient

The adjustment of the quantity of network position vectors of the fishing network M constitutes a fundamental factor to achieve the efficiency of the algorithm. In the definition of an adequate value or range for M, we have to consider the following: a big fishing network (of many position

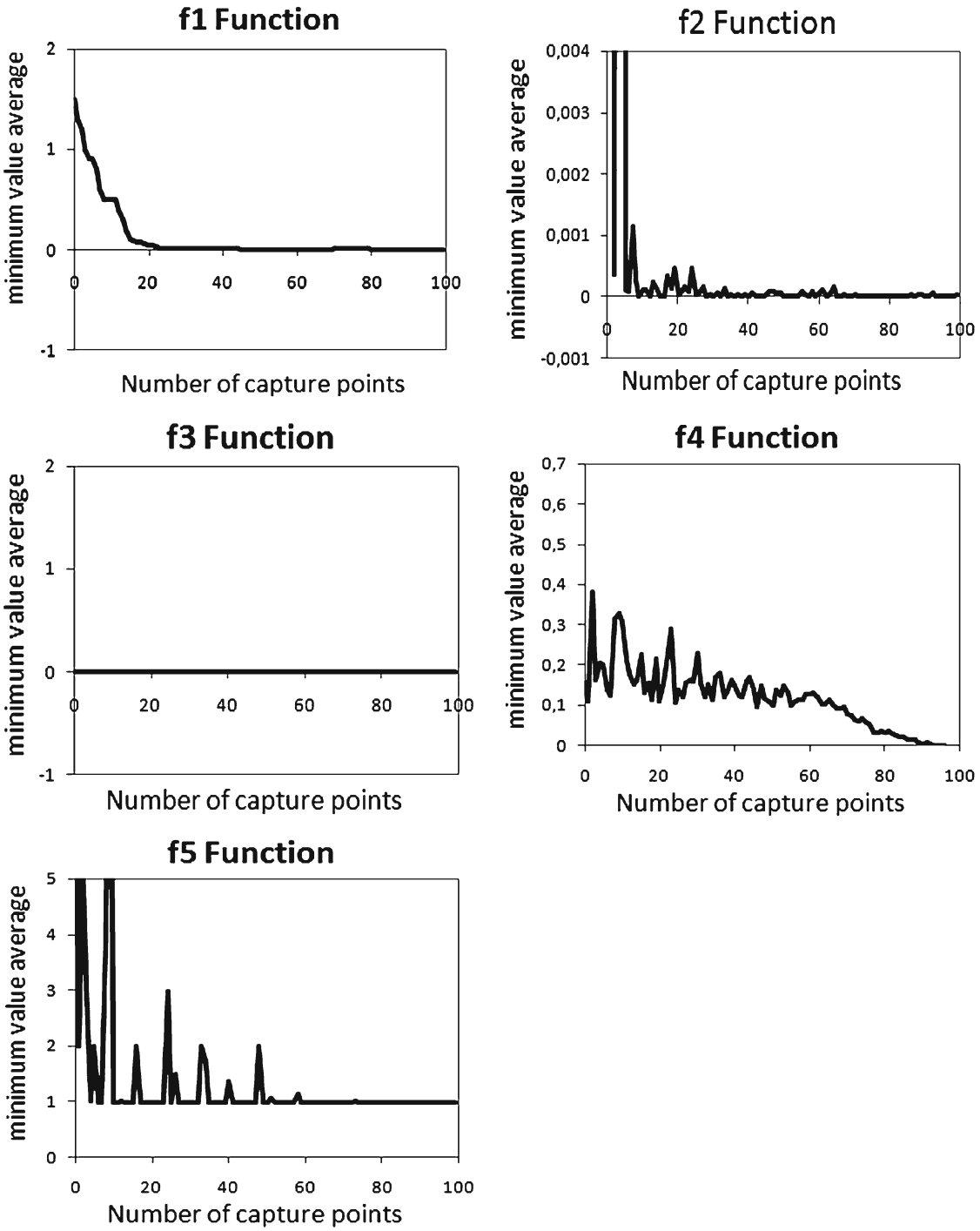


Fig. 3 Results for De Jong's five function test, considering the capture points

vectors) guarantees to us a deep exploration, but increases considerably the computational burden because it has to make more calls to the objective function. On the other hand, if the fishing network is small, the fast convergency of the algorithm does not guarantee a good solution for the problem.

In this sense, it is very important to know the type of the function and the problem we are facing and to establish an effective relation with another essential parameter, the width coefficient c.

If c is too small, the search can be slow and expensive; but if c has a big value and M is also small, the algorithm can end

Table 2 Average values found for different numbers of iterations and throws

f	Throws	Iterations				
		100	200	300	400	500
f_1	3	1.11×10^{-3}	1.52×10^{-3}	1.00×10^{-3}	8.18×10^{-4}	3.15×10^{-4}
	5	1.10×10^{-4}	5.27×10^{-4}	4.12×10^{-5}	8.36×10^{-6}	4.25×10^{-7}
	10	9.21×10^{-4}	8.97×10^{-4}	6.54×10^{-5}	5.66×10^{-6}	6.54×10^{-7}
	15	5.25×10^{-4}	6.54×10^{-5}	7.58×10^{-4}	6.58×10^{-6}	3.35×10^{-6}
	20	3.15×10^{-4}	7.45×10^{-5}	2.22×10^{-5}	1.01×10^{-5}	9.02×10^{-7}
	25	4.96×10^{-4}	9.87×10^{-4}	6.35×10^{-4}	5.47×10^{-5}	2.55×10^{-6}
f_2	3	4.16×10^{-6}	9.99×10^{-6}	8.15×10^{-6}	1.29×10^{-7}	5.65×10^{-7}
	5	1.36×10^{-7}	4.55×10^{-7}	7.33×10^{-7}	8.24×10^{-8}	4.27×10^{-8}
	10	5.42×10^{-6}	6.49×10^{-7}	6.32×10^{-6}	7.69×10^{-6}	8.57×10^{-7}
	15	3.25×10^{-7}	6.54×10^{-7}	4.25×10^{-7}	4.36×10^{-8}	8.73×10^{-8}
	20	6.14×10^{-7}	2.11×10^{-7}	9.68×10^{-7}	9.58×10^{-8}	1.68×10^{-8}
	25	1.20×10^{-6}	3.48×10^{-6}	4.57×10^{-6}	4.77×10^{-7}	9.47×10^{-7}
f_3	3	0	0	0	0	0
	5	0	0	0	0	0
	10	0	0	0	0	0
	15	0	0	0	0	0
	20	0	0	0	0	0
	25	0	0	0	0	0
f_4	3	1.27×10^{-2}	6.74×10^{-3}	9.00×10^{-2}	5.44×10^{-3}	7.82×10^{-3}
	5	3.55×10^{-3}	4.65×10^{-2}	7.33×10^{-3}	1.04×10^{-2}	8.14×10^{-3}
	10	9.58×10^{-2}	2.11×10^{-3}	1.54×10^{-2}	2.62×10^{-3}	1.34×10^{-2}
	15	1.82×10^{-2}	7.05×10^{-2}	9.87×10^{-3}	1.87×10^{-2}	1.24×10^{-3}
	20	5.68×10^{-3}	9.88×10^{-2}	3.77×10^{-3}	9.52×10^{-3}	1.99×10^{-2}
	25	3.65×10^{-3}	8.47×10^{-2}	2.03×10^{-2}	2.41×10^{-3}	7.44×10^{-3}
f_5	3	1.0582	0.9980	0.9980	0.9980	0.9980
	5	0.9980	0.9980	0.9980	0.9980	0.9980
	10	0.9980	0.9980	0.9980	0.9980	0.9980
	15	0.9980	0.9980	0.9980	0.9980	0.9980
	20	0.9980	0.9980	0.9980	0.9980	0.9980
	25	0.9980	0.9980	0.9980	0.9980	0.9980

up in an early and imprecise convergency because of the lack of exploration in zones where an optimum might be found.

On the other hand, FSP applies a strategy to update the width coefficient c, but in this section, we will focus in determining the best values of c and M for the De Jong's functions test, it means, not incorporating any improvement strategy.

In Table 3, we show the recommended values for each parameter of the algorithm and each test function.

Furthermore, if we analyze the searching region (S) corresponding to each function and the values of c, we can appreciate that with an increase in the searching region, the values of c tend to increase too; it is not the case for M.

4 Experimentation and evaluation

In this section, we present the experimental results of the evaluation stage of our proposal in terms of successful runs (convergence on global minimum values) and time consumption. We carried out comparisons considering other metaheuristics and interesting proposals referenced in the literature. For evaluation, we have used a set of benchmark functions broadly used to test the behavior and convergence of heuristic methods. Finally, we show and analyze the results of the achieved performance by different methods.

Table 3 Recommended parameters for the test functions

Function	Parameters				
	T	N	L	M	c
f_1	500	25	5	200	20.0
f_2	400	80	5	80	0.10
f_3	100	10	3	30	0.70
f_4	100	100	3	100	0.05
f_5	100	80	5	80	10.0

4.1 Benchmark functions

In order to evaluate the novel metaheuristic, a test suite of benchmark functions previously introduced by Molga and Smutnicki [13] was used (see Table 4). The ranges of search spaces, dimensionalities, and global minimum function values (ideal values) are also included in Table 4. In each case, n is the dimension size of the function, $f_{\min}$ is its ideal value, and $S \subset R^n$ is the search space. The problem set contains a diverse set of problems, including unimodal as well as multimodal functions, and functions with correlated and uncorrelated variables.

Functions f_1–f_3 are unimodal. Functions f_4 and f_5 are multimodal functions where the number of local minima increases exponentially with the problem dimension. Functions f_6–f_{11} are low-dimensional functions, which have only a few local minima.

4.2 Selected metaheuristics for comparison

With the intention of comparing FSP with other heuristic procedures, we selected GRASP, DE, and PSO methods. This selection was carried out with the goal of evaluating the behavior of the proposed method with similar algorithms (GRASP) and with other completely different ones (DE and PSO), demonstrating the potentialities of FSP in the solution of optimization problems.

GRASP [8] is a multistart metaheuristic in which each iteration consists basically of two phases: construction and local search. One possible shortcoming of the standard GRASP framework is the independence of its iterations, i.e., the fact that it does not learn from the history of solutions found in previous iterations. This is so because the basic algorithm discards information about any encountered solution that does not improve the function value. Information gathered from good solutions can be used to implement memory-based procedures to influence the construction phase, by modifying the selection probabilities associated with each element of the restricted candidate list.

PSO is a population-based stochastic optimization technique developed by Eberhart and Kennedy [7] in 1995, inspired by social behavior of bird flocking or fish schooling. The main advantages are as follows: insensitive to scaling of design variables, simple implementation, easily parallelized for concurrent processing, derivative free, very few algorithm parameters, and very efficient global search algorithm. In spite of this, it presents the disadvantage of a slow convergence in refined search stage (weak local search ability).

DE algorithm has been introduced by Storn and Price [14]. DE is a method of optimization belonging to the category of evolutionary computation. It maintains a population of solution candidates, to which a mutation and recombination procedure to generate new individuals is applied. The new individuals are chosen according to the value of its performance function. The main characteristic of DE is the use of test vectors, which compete with the current population individuals in order to survive. One of their biggest advantages is that it overcomes the standard drawbacks of the genetic algorithms: premature convergence and lack of good local search ability. However, DE sometimes has a limited ability to move its population large distances across the search space and would have to face stagnation.

Although FSP not always outperforms the other methods, it solves some of their inconveniences, e.g., the independence of iterations in GRASP, the weak capacity of local search of PSO, and the limited ability to face stagnation in DE.

4.3 Evaluation

The algorithms used for comparison are GRASP [8], PSO with inertia weight and constriction factor (PSO-w-cf) [15, 16], and differential evolution (DE) algorithm [14]. In all experiments, a total of 30 runs are made. The experiment results are listed in Table 5, where Func. = Functions, Algo. = Algorithms, best stands for the best function value over 30 runs, mean indicates the mean best function values, and time stands for the average CPU time (in seconds) consumed within the fixed number of generations. Succ. Runs stand for the success number over 30 runs. All results are reported with a precision level of 1e−02.

The initial population is generated uniformly and randomly in the range as specified in Table 4. The parameters of (PSO-w-cf) are as follows: learning rate $c_1 = c_2 = 2$, inertia weight linearly decreased from 0.9 to 0.4 with run time increasing, constriction factor $\chi = 0.729844$, and the maximum velocity $v_{\max}$ is set at 20 % of the dynamic range of the variable on each dimension. The parameters of DE are mutation $f = 0.75$, recombination cr $= 0.5$, number of objective vectors NVO $= 6$, and strategy $s = 4$.

The parameters of FSP are as follows: the number of iterations and the capture points are fixed between 80 and 100, number of launchings from 3 to 5, the number of points of the fishing network in 100, and the width coefficient was

Table 4 The employed benchmark functions

Functions	n	S	$f_{\min}$
$f_1(\vec{x}) = \sum_{i=1}^{n} x_i^2$	5	$[-5.12, 5.12]^n$	$f_1(\vec{0}) = 0$
$f_2(\vec{x}) = \sum_{i=1}^{n} \left(i \cdot x_i^2\right)$	5	$[-5.12, 5.12]^n$	$f_2(\vec{0}) = 0$
$f_3(\vec{x}) = \sum_{i=1}^{n} \sum_{j=1}^{i} x_j^2$	2	$[-65.536, 65.536]^n$	$f_3(\vec{0}) = 0$
$f_4(\vec{x}) = \sum_{i=1}^{n} \left[-x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)\right]$	2	$[-500, 500]^n$	$f_4(\vec{420.9687}) = -418.9829n$
$f_5(\vec{x}) = \sum_{i=1}^{n} \lvert x_i \rvert^{i+1}$	2	$[-1.0, 1.0]^n$	$f_5(\vec{0}) = 0$
$f_6(x_1, x_2) = \left(x_2 - x_1^2\right)^2 + (1 - x_1)^2$	2	$[-1.9, 2.0]$	$f_6(1, 1) = 0$
$f_7(x, y) = (x - 13 + ((5 - y)\, y - 2)\, y)^2 + (x - 29 + ((y + 1)\, y - 14)\, y)^2$	2	$[-1, 12]$	$f_7(5, 4) = 0,\ f_7(11.41, -0.8986) = 48.9842$
$f_8(x_1, x_2) = a\left(x_2 - bx_1^2 + cx_1 - d\right)^2 + e(1 - f)\cos(x_1) + e$	2	$[-\pi, 13]$	$f_8(-\pi, 12.275) = 0.397887,$ $f_8(\pi, 2.275) = 0.397887,$ $f_8(9.42478, 2.475) = 0.397887$
$f_9(x, y) = (1.5 - x(1 - y))^2 + \left(2.25 - x\left(1 - y^2\right)\right)^2 + \left(2.625 - x\left(1 - y^3\right)\right)^2$	2	$[0.0, 5.0]$	$f_9(3, 0.5) = 0$
$f_{10}(x_1, x_2) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + \left(-4 + 4x_2^2\right)x_2^2$	2	$x_1[-3, 3],\ x_2[-2, 2]$	$f_{10}(-0.0898, 0.7126) = -1.0316,$ $f_{10}(0.0898, -0.7126) = -1.0316$
$f_{11}(x_1, x_2) = \left[1 + (x_1 + x_2 + 1)^2\left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right] \cdot \left[30 + (2x_1 - 3x_2)^2\left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$	2	$[-2, 2]$	$f_11\,(0, -1) = 3$

Table 5 Performance of the algorithms

Func.	Algo.	Best	Mean	Time	Succ. Runs
f_1 (De Jong's)	DE	1.7328e−02	9.5487e−02	2.58	05
	PSO-w-cf	8.3241e−13	3.6998e−11	**0.98**	**30**
	GRASP	1.4805e−01	1.17728420	76.1	00
	FSP	1.4058e−02	7.7205e−02	2.21	07
f_2 (Axis parallel hyper-ellipsoid)	DE	1.6242e−02	8.8914e−02	9.21	04
	PSO-w-cf	6.1828e−10	9.1778e−09	**1.04**	**30**
	GRASP	0.42167256	1.67167256	85.3	00
	FSP	1.7272e−02	9.3354e−02	9.91	02
f_3 (Rotated hyper-ellipsoid)	DE	6.1984e−05	7.1452e−05	0.06	**30**
	PSO-w-cf	1.2611e−04	2.2155e−04	**0.03**	**30**
	GRASP	4.1835e−03	3.1987e−02	1.51	00
	FSP	1.7315e−03	1.7956e−02	0.06	08
f_4 (Schwefel's)	DE	−837.96586	−837.36187	**0.06**	02
	PSO-w-cf	−837.96586	−837.12575	0.08	05
	GRASP	−837.91124	−837.21545	7.74	00
	FSP	−837.96581	−837.61520	0.08	**06**
f_5 (Sum of different power functions)	DE	1.7257e−05	8.5222e−04	**0.01**	18
	PSO-w-cf	1.9099e−05	6.6587e−04	**0.01**	21
	GRASP	2.9311e−04	3.1533e−03	0.06	00
	FSP	8.6995e−06	1.9112e−05	**0.01**	**30**
f_6 (Rosenbrock or Banana)	DE	8.4157e−03	2.4723e−02	**0.01**	00
	PSO-w-cf	1.6448e−05	5.3687e−04	**0.01**	**16**
	GRASP	5.7065e−04	1.2365e−03	0.05	00
	FSP	1.5161e−05	2.8734e−04	**0.01**	14
f_7 (Freudenstein and Roth)	DE	5.1626e−04	1.1234e−03	0.07	**30**
	PSO-w-cf	1.0961e−04	3.5698e−03	**0.06**	23
	GRASP	1.9228e−03	1.4589e−02	1.09	11
	FSP	3.0940e−03	1.1090e−03	**0.06**	**30**
f_8 (Branins's)	DE	0.39781959	0.40254578	**0.03**	01
	PSO-w-cf	0.39788737	0.39998751	**0.03**	**25**
	GRASP	0.39782849	0.41587485	0.67	01
	FSP	0.39787495	0.39954781	**0.03**	12
f_9 (Beale)	DE	1.3211e−03	3.6587e−02	**0.03**	18
	PSO-w-cf	1.8051e−05	2.7624e−04	**0.03**	**30**
	GRASP	3.3778e−04	9.5874e−03	1.07	20
	FSP	5.5255e−04	1.4423e−03	**0.03**	**30**
f_{10} (Six-hump camel back)	DE	−1.0316284	−1.0375987	**0.04**	**08**
	PSO-w-cf	−1.0316625	−1.0335411	**0.04**	01
	GRASP	−1.0310172	−1.0405474	0.99	00
	FSP	−1.0316738	−1.0381120	**0.04**	02
f_{11} (Goldstein-Price's)	DE	3.00000082	3.00054785	**0.03**	**30**
	PSO-w-cf	3.00711378	3.01965874	**0.03**	21
	GRASP	3.00664548	3.09925453	0.71	17
	FSP	3.00285013	3.01125478	**0.03**	23

The bold values represent the best values obtained by the four algorithms for the corresponding function

adjusted by the own method beginning in 0.05, according to the characteristics of the functions.

All algorithms are run on a PC with Intel(R) Core(TM) i5 CPU 2.53 GHz.

The results of 30 independent runs for benchmark functions f_1–f_{11} are summarized in Table 5. From Table 5, FSP is successful over all the 30 runs for f_5, f_7, and f_9. For f_4, it is successful in only 20 % of all runs, but it outperforms all other algorithms. For the functions 1, 6, and 8, FSP always outperforms GRASP and DE methods in successful runs. For f_1, FSP is successful in 23 % of the runs and has less time consumption than GRASP and DE. For f_6 and f_8, FSP is successful in 46 and 40 % of the runs, respectively, for both cases, and has less time consumption than GRASP and the same as DE. For f_{10} and f_{11}, FSP is successful in 46 and 40 % runs, respectively, and is better than GRASP and PSO-w-cf methods to achieve the desired accuracy. For this function, it also has less time consumption than GRASP and the same as PSO-w-cf. In f_2 and f_3, FSP overcomes GRASP in computational time and successful runs.

The results with benchmark functions allow us to conclude that FSP is suitable for solving optimization problems for unimodal and multimodal functions with satisfactory (second best) convergence ability. Compared with classic GRASP metaheuristic, FSP has better search ability and less time consumption, expressed in terms of successful runs and total runtime, respectively, for the benchmark functions. Moreover, FSP has a competitive performance for all benchmark functions, compared with PSO-w-cf and DE.

5 Future research

In this section, considering the initial conceptions of this work, the authors propose four extensions of the FSP that can be discussed, implemented, and evaluated in future works.

In extension #1, the fisherman updates the position of an specific capture point i just when one of the fishing network's points improves the function value in p_i. Under this idea, if after a throwing of the net, the function value in p_i does not improve, no other throw is made in the corresponding capture point until the next iteration.

After each time, the fishing network is thrown and the function value in p_i for the ith capture point is improved; the amount of fishing network's points and the width coefficient values are decreased by a specific factor (0.99). In the next iteration of the method, the values of these two parameters are reset to their default values.

In the beginning, extension #2 is similar to extension #1, because the fisherman does not update the position of a capture point i unless one of the fishing network points (threw from i) improves the function value in p_i. While the capture point position keeps improving, fishing network is thrown continuously and the width coefficient value decreases according to the extension #1 factor.

After each iteration, those capture points whose position did not improve are restarted to a new position. This restart can be random or guided to those unexplored zones.

In extension #3, in each capture point, all throws of the fishing network are made. Different to other extensions, the amount of throws is not conditioned to position improvements. In this case, each time the fishing network is thrown in a specific capture point, the position of this point is always updated with the best points obtained so far by the fishing network.

This idea is based on the intuition that a bigger fish (optimal solution) can be in zones where small fishes live (not so good solutions), those that serve as food to it.

In extension #4, we start by dividing the fishing zone (searching space) in subregions and sending a fisherman to each of them. Each fisherman can apply indistinctly his own extension. This cooperative of fishermen shares a common memory where the best position (g_{best}) reached by any of the fishermen is stored.

Even though parallelism is not yet systematically used to speed up or to improve the effectiveness of metaheuristics, parallel implementations are very robust and abound in the literature; see, e.g., Cung et al. [17] for a survey.

In general, each extension has its advantages and disadvantages. In extension #1, the fisherman tries to delimit the best position with abrupt steps, but he can get trapped in a local minimum. In spite of the similitude with extension #1, in extension #2, the fisherman uses the restart of the capture points as the main strategy to avoid an early convergency in local minimums. On the other hand, extension #3 is conceived with the idea that the trajectory, passing through not so promising zones, could take us to the best capture zones, which would be inaccessible otherwise. In extension #4, we suggest parallel implementations to enhance the robustness of the method.

Finally, we conclude that the selection and application of the mentioned FSP extensions, and others, depend on the characteristics and situations where future optimization problems can arise. We are currently working in these extensions of the basic model.

6 Conclusions

In this paper, we have proposed a new metaheuristic called Fisherman Search Procedure. The search algorithm explores new solutions using a combination of guided and local search. Our metaheuristic was designed with the purpose of developing good solutions for a wide variety of global optimization problems.

The main advantages of FSP are the following: it provides an explicit description of the model-based idea, easy implementation, and the conception of this metaheuristic allows it to be applied to a large number of classical global optimization problems, as well as to those that arise in real-world situations in different areas of science, technology, and business.

Finally, we conclude that FSP has better search ability and less time consumption, expressed in terms of successful runs and total runtime, respectively, for the benchmark functions, compared with classic GRASP metaheuristic. In 90 % of the cases, FSP is located among the two better results as for successful runs. On the other hand, with regard to time consumption, FSP shows similar results to PSO and DE, achieving the best and second best results for 82 % of the test functions.

As future work, we plan to analyze the quality of the solution with other benchmark test functions and validate the four proposed extensions under different conditions and optimization problems.

Acknowledgments This paper has been supported by the Spanish Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía and the Ministerio de Ciencia e Innovación under the projects P09-TIC-4526 and TIN2011-28538-C02-02, respectively.

References

1. Glover, F., Laguna, M.: Tabu search. Kluwer Academic Publishers, Dordrecht (1997). (ISBN-10: 079239965X)
2. Kirkpartick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
3. Calegari, P., Coray, G., Hertz, A., Kobler, D., Kuonen, P.: A taxonomy of evolutionary algorithms in combinatorial optimization. J. Heuristics **5**, 145–158 (1999)
4. Moscato, P.: Memetic algorithms: a short introduction. In: Corne, D., Dorigo, M., Glover, F. (eds.) New ideas in optimization, pp. 219–234. McGraw-Hill, London (1999)
5. Glover, F.: Scatter search and path relinking. In: Corne, D., Dorigo, M., Glover, F. (eds.) New ideas in optimization, pp. 297–316. McGraw-Hill, London (1999)
6. Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the travelling salesman problem. IEEE Trans. Evol. Comput. **1**(1), 53–66 (1997)
7. Eberhart, R.C., Kennedy J.: A new optimizer using particles swarm theory. In: Proceedings of Sixth International Symposium on Micro Machine and Human Science, pp. 39–43. Nagoya, Japan (1995)
8. Resende, M.G.C., Ribeiro, C.C.: Greedy randomized adaptive search procedures. Handbook of metaheuristics. Kluwer Academic Publishers, Dordrecht (2003)
9. Stützle, T.: Local search algorithms for combinatorial problems—analysis, algorithms and new applications. DISKI—Dissertationen zur Künstliken Intelligenz, Infix (1999)
10. Stützle, T.: Iterated local search for the quadratic assignment problem. Technical report, TU Darmstadt (1999)
11. Hansen, P., Mladenovic, N.: An introduction to variable neighborhood search. Meta-heuristics: advances and trends in local search paradigms for optimization. Kluwer Academic Publishers, Dordrecht (1999)
12. Jong, K.D.: Analysis of the behavior of a class of genetic adaptive systems. PhD thesis, The University of Michigan (1975)
13. Molga, M., Smutnicki, C.: Test functions for optimization needs, http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf
14. Storn, R., Price, K.: Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report TR-95-012. International Computer Science Institute, Berkeley (1995)
15. Clerc, M., Kennedy, J.: The particle swarm—explosion, stability, and convergence in a multidimensional complex space. In: IEEE Transactions on Evolutionary Computation, pp. 58–73. IEEE Press, New York (2002)
16. Shi, Y., Eberhart, R.: A modified particle swarm optimizer. In: IEEE World Congress on Computational Intelligence, pp. 69–73. IEEE Press, New York (1998)
17. Cung, V-D., Martins, S.L., Ribeiro, C.C., Roucairol, C.: Strategies for the parallel implementation of metaheuristics. Essays and Surveys in Metaheuristics, pp. 263–308. Kluwer Academic Publisher (2002)