Prey-Predator Algorithm: A New Metaheuristic Algorithm for Optimization Problems

Article *in* International Journal of Information Technology and Decision Making · December 2015

DOI: 10.1142/S021962201450031X

Surafel Luleseged Tilahun
Addis Ababa Science and Technology University
47 PUBLICATIONS **866** CITATIONS
SEE PROFILE

Hong Choon Ong
Universiti Sains Malaysia
65 PUBLICATIONS **681** CITATIONS
SEE PROFILE

International Journal of Information Technology & Decision Making
Vol. 13 (2014)
© World Scientific Publishing Company
DOI: 10.1142/S021962201450031X

PREY-PREDATOR ALGORITHM: A NEW METAHEURISTIC ALGORITHM FOR OPTIMIZATION PROBLEMS

SURAFEL LULESEGED TILAHUN

Computational Science Program, Faculty of Science
Addis Ababa University
1176, Addis Ababa, Ethiopia
surafelaau@yahoo.com

HONG CHOON ONG

School of Mathematical Sciences
UniversitiSains Malaysia
11800, USM, Pulau Pinang, Malaysia
hcong@cs.usm.my

Published 12 December 2013

Nature-inspired optimization algorithms have become useful in solving difficult optimization problems in different disciplines. Since the introduction of evolutionary algorithms several studies have been conducted on the development of metaheuristic optimization algorithms. Most of these algorithms are inspired by biological phenomenon. In this paper, we introduce a new algorithm inspired by prey-predator interaction of animals. In the algorithm randomly generated solutions are assigned as a predator and preys depending on their performance on the objective function. Their performance can be expressed numerically and is called the survival value. A prey will run towards the pack of preys with better surviving values and away from the predator. The predator chases the prey with the smallest survival value. However, the best prey or the prey with the best survival value performs a local search. Hence the best prey focuses fully on exploitation while the other solution members focus on the exploration of the solution space. The algorithm is tested on selected well-known test problems and a comparison is also done between our algorithm, genetic algorithm and particle swarm optimization. From the simulation result, it is shown that on the selected test problems prey-predator algorithm performs better in achieving the optimal value.

Keywords: Metaheuristic algorithm; prey-predator algorithm (PPA); optimization; bio-inspired algorithms.

1991 Mathematics Subject Classification: 90-08, 78M50, 90B99, 90B50

1. Introduction

Formulating a problem mathematically as an optimization problem in order to solve it using scientific reasoning has become useful in many applications.[1–3] The formulated problem can be classified into different categories depending on the objective function and the solution space. Linear or nonlinear, convex or nonconvex and constrained or unconstrained problems are very popular classifications. In reality there are many hard optimization problems which need a special attention other than the known deterministic solution methods. Metaheuristic algorithms are perhaps good solution methods to solve hard optimization problems.[4] Metaheuristic algorithms do not guarantee optimality but they will give a reasonable solution acceptable by the decision maker. The use of metaheuristic solution algorithms for optimization problems can be traced back to the 1960's. In the 1960's, evolutionary algorithm was developed by John Holland.[5,6] Since the introduction of evolutionary algorithm many metaheuristic algorithms have been developed. Most of these algorithms are inspired by natural phenomenon. For instance simulated annealing was developed in 1983,[7] inspired by annealing process in material processing when a metal cools and freezes. In 1992, Ant colony optimization was developed mimicking the behavior of ants.[8] Particle swarm optimization (PSO) was also developed in the same decade, in 1995.[9] Generally the 1980's and 1990's were exciting times for metaheuristic algorithm development.[6] Recently, a number of algorithms were also introduced. These include Firefly algorithm,[6] Cuckoo search[6] and Human evolutionary model approach.[10] Apart from introducing new algorithms, some of these algorithms are also combined with other algorithms or methods to improve the performance in solving a particular problem.[11–15] Each of these metaheuristic algorithms has its own strengths and weaknesses. These algorithms are tested and used on problems from different disciplines.[16–20]

An Ecosystem is a biological environment which contains the living things and the nonliving things together. Natural ecosystem has evolved with all the components and the living things together. Learning from nature is perhaps a good idea in solving problems. Nature takes care of things without being told what to do and how to do it. Almost all metaheuristic algorithms are inspired by phenomenon from the ecosystem. In the ecosystem, living things interact with one another in different ways. Kerbs lists the interactions in six different categories.[21] Competition occurs when two species compete in using the same resource. Herbivory is a situation where an animal eats plants. Predation is where an animal eats another animal. The other interactions, listed by Kerbs, are parasitism, disease and mutualism. Some books refer herbivory as a predation, and they classify predation as herbivory predation and carnivory predation.[22] Taking a close look on the carnivory predation in which the predator chases the prey is inspiring to see that the prey has to be alert and outrun the predator or in some cases to find a hideout to hide itself, as in the case of rabbits. These animals have long evolved. This means that the prey has managed to find a hideout or in general manage to survive, and the predator has also managed to hunt and eat the prey which is not fit enough to escape or is unfortunate.

Inspired by this prey-predator interaction we introduce a new algorithm for optimization problems. In the algorithm, randomly generated solutions are assigned as a predator and preys depending on their performance in the objective function.

This performance is called survival value. A solution with lowest survival value is assigned as the predator and the other solutions as preys. Then the predator chases the weakest prey and the prey will run away from the predator and try to follow the pack of preys with better survival values. The prey with best survival value is considered as a prey which has found the hideout and will do a local search only. The reassignment of a predator and preys will be done in each iteration. The prey-predator algorithm (PPA) is also compared with genetic algorithm (GA) and PSO, on selected five bench mark problems of different properties. From the comparison and simulation results, it is shown that PPA performs better on the selected test problems and is a promising algorithm for solving optimization problems.

In the next section, basic concepts will be discussed followed by a detailed presentation on the algorithm in Sec. 3. Section 4 is devoted to the comparison study between PPA, GA and PSO. In Sec. 5, simulation results will be discussed and finally the conclusion and some possible future extensions.

2. Preliminaries

2.1. *Optimization problems*

An optimization problem is a problem of finding an element from a set called feasible or constraint set which will optimize a given objective function. To optimize can be either to maximize or minimize. Without loss of generality in this paper we consider a maximization problem as given below:

$$\max_{x \in S \subseteq \Re^n} f(x), \tag{2.1}$$

where $f(x)$ is the objective function; x is a decision variable and S is the feasible region. x^* is said to be a global optimal solution for the maximization problem in Eq. (2.1) if and only if $x^* \in S$ and $f(x^*) \geq f(x)$ for all x in S. A local maximizer is an element of S which does better in its neighborhood region. This means x^* is local maximizer of $f(x)$ if and only if $x^* \in S$ and $f(x^*) \geq f(x)$ for all x in $\{x \in S : \|x - x^*\| < \delta, \text{ for some } \delta > 0\}$.[23]

2.2. *Prey-predator interaction*

Predation can be defined as the consumption of a living organism by another living organism.[24] There are different types of prey-predator interactions based on the type of prey and how the predator consumes its prey. Our focus is on carnivorous predation. Different types of animals are involved in carnivorous predation. Predators as used in this paper refer to those predators that run after their prey. Capturing the prey is a challenging task; hence predators have to be fast-moving. For instance cheetah is the fastest predator and can run with a speed of 70–75 miles per hour for about 460 m.[25] Lions are also fast hunters with a speed of up to 50 miles per hour for short distances.[26] Even though predators have high speeds to hunt their prey; most preys have the advantage of running fast for longer period. Pronghorn antelope can run 40–57 miles per hour for long distances.[27] Gazelle is also another prey which runs with a high speed of 40–47 miles per hour and its quick turns confuse the predator.[28]

Generally, in hunting, a predator must find, recognize and capture the prey. As most predators can outrun their prey for a short distance only they have to find and recognize the prey first. Following that it has to go nearer while not being noticed by the prey. When the distance is suitable to sprint and catch one of the nearest preys, it will run fast towards the prey. Since the stronger preys and those that are far can outrun the predator easily, it has to identify the easy target which is nearer and slower. The predator can hunt either in group like the lions, or individually like foxes. At the same time the preys will try to escape or protect themselves. They can do this either by running away, hiding or by struggling. Musk oxen are good examples for struggling. They are known for thwarting wolf predation by pressing closely together in a circle with their horns facing outwards.[29] However, most preys run as fast as they can away from the predator. Perhaps, antelope, gazelle, zebra and rabbit are good examples. Being in a group is also advantageous for some preys. For instance buffalos have the advantage of survival if they are in a group. And in many cases the preys will prefer to run in a pack together which is mostly not good for hunting.

Inspired by such prey-predator relationship we discuss our proposed PPA in the next section.

3. Prey-Predator Algorithm (PPA)

The PPA is a metaheuristic optimization algorithm inspired by prey-predator interaction of animals. In the algorithm initial set of feasible solutions are generated. Each solution member, x_i, will be assigned with a survival value, $\mathrm{SV}(x_i)$, which depends on the encoded objective function. For the maximization problem, the survival value of the solutions should be directly proportional to the objective function.

$$\mathrm{SV}(x) \propto f(x). \tag{3.1}$$

This can be considered as how far the prey is from the predator or the strength of a prey to outrun the predator. After the survival value for each solution member is computed, a member with the least survival value will be assigned as a predator and the rest as preys. Once the preys and the predator are assigned, the preys need to run away from the predator and try to follow the better preys in terms of survival values or find a hiding place at the same time. Exploration or diversification is done by the preys and the predator, while the predator is hunting and the prey tries to run away from the predator. Exploitation or intensification is done by the preys, especially by the prey with best surviving value, using a local search. The prey with best surviving value is considered as the one who has found the hiding place and is not affected by the predator, but rather will do a local search only to improve the survival value. However, the other preys will follow the prey population with better surviving values. There are two basic issues when we talk about the movement of preys or the predator; the first is the direction, a unit vector, in which the movement is going to take place and the second is the step length, which determines how far on that direction should a prey or the predator moves.

In generating a movement direction for a prey, x_i, we consider the location of those preys with better surviving values. By a given follow up probability, the prey tends to follow the mass with better surviving value and do a local search as well. If the follow-up probability is not met it will move randomly away from the predator. In the first case, in following preys with better survival values and doing a local search, it is necessary to identify those preys which have better survival values, say $\{x_1, x_2, \ldots, x_p\}$. Naturally a prey tries to get itself with the nearest pack of preys. Hence the direction depends not only on the survival values but also on the distances between x_i and the location of preys with better survival values. Then the direction for x_i is given by:

$$y_i = \sum_{j=1}^{p} \frac{\eta^{\mathrm{SV}(x_j)}(x_j - x_i)}{r_{ij}}, \tag{3.2}$$

where $\eta \geq 1$, r_{ij} is the distance between x_i and x_j. In this study, the Euclidean distance is used to compute the distance between vector solutions; $r_{ij} = \|x_i - x_j\|$ and it is also possible to use γr_{ij}^{α} instead of r_{ij}, for $\gamma > 0$ and $\alpha \geq 0$ depends on how much we need it to depend on the distance between the given prey and other preys with better survival values. Assigning large values for parameters γ and α, means making the direction highly dependent on the distance between them and assigning a very small value means making the direction less dependent on the distance. For instance taking $\alpha = 0$ makes y_i to be independent of the distance. In this paper we take both γ and α to be 1. Similarly, η shows the degree in which the direction is affected by the survival value of better solutions. $\eta^{\mathrm{SV}(x_j)}$ can also be expressed using $e^{(\mathrm{SV}(x_j))^{\tau}}$, for appropriate choice of τ. For practical case, to eliminate the singularity case in which $r_{ij} = 0$ we can modify the formula for y_i as follows:

$$y_i = \sum_{j=1}^{p} e^{(\mathrm{SV}(x_j))^{\tau} - r_{ij}}(x_j - x_i). \tag{3.3}$$

The local search is done by generating random directions and checking if there is any direction which increases the survival value if the solution moves in that direction. Let y_l be the best direction for the local search. If such a direction is not found in the randomly generated directions, we set y_l to be a zero vector.

Furthermore, if the follow-up probability is not met, the prey, x_i, will run in a random direction away from the predator. This is done by generating a random direction, y_r and comparing y_r and $-y_r$. To identify which direction will take the prey away from the predator we calculate the distance between the predator and the prey by assuming that the prey moves in these directions. To do so we just compute the distance between the new location of the prey and the location of the predator as in Eqs. (3.4) and (3.5).

$$d_1 = \|x_{\text{predator}} - (x_i + y_r)\|, \tag{3.4}$$

$$d_2 = \|x_{\text{predator}} - (x_i - y_r)\|. \tag{3.5}$$

If $d_1 < d_2$ we take $-y_r$ to be the random direction otherwise we take y_r to be the random direction.

The other case for a prey x_i is the situation where there is no other better prey with higher survival value; let us call that prey *the best prey*. In such a case the best prey will do a local search, by generating random directions and check if there is a direction in which the survival value increases. If there is a direction, y_l, in which the survival value increases then the prey will go in that direction if not, it will stay in its current position.

The next issue is the step length. The prey which is far from the predator will not run as fast as the prey which is near and chased by the predator. Hence, the prey moves with a speed inversely proportional to its relative surviving value. Let the maximum step length, a prey can run be $\lambda_{\max}$. Then the step length of a prey, x_i, can be expressed as follows for positive ω and β:

$$\lambda = \frac{\lambda_{\max}\varepsilon_1}{\beta\,|\mathrm{SV}(x_i) - \mathrm{SV}(x_{\text{predator}})|^{\omega}}, \tag{3.6}$$

where ε_1 is a random number, from a uniform probability distribution, between 0 and 1. It is preferable to express it using Eq. (3.7), to omit a singularity case.

$$\lambda = \frac{\lambda_{\max}\varepsilon_1}{e^{\beta\,|\mathrm{SV}(x_i) - \mathrm{SV}(x_{\text{predator}})|^{\omega}}}. \tag{3.7}$$

The parameter β determines how the step length depends on the relative surviving value. $\beta = 0$ means the prey's step length does not depend on the surviving value of x_i. Assigning a large value for β will make the step length highly affected by the surviving value. This in turn will make some preys not to move or move with a very small step length. This will again affect the exploration property of the algorithm. Similarly, ω has the same effect; taking $\omega = 0$ makes it independent of the distance and taking larger number makes it highly dependent on the surviving value. The assignment of these two parameters depends on the survival value, $\lambda_{\max}$ and how big the search space is. In this paper we set both of these two parameters to be one.

Generalizing all the points discussed, the movement of the prey, excluding the best prey can be summarized as in Eq. (3.8).

$$x_i \leftarrow x_i + l_{\max}(\varepsilon_2)\left(\frac{y_i}{\|y_i\|}\right) + \varepsilon_3\left(\frac{y_r}{\|y_r\|}\right), \quad \text{where } l_{\max} = \frac{\lambda_{\max}}{e^{\beta|\mathrm{SV}(x_i) - \mathrm{SV}(x_{\text{predator}})|}}. \tag{3.8}$$

For the case of the best prey:

$$x_i \leftarrow x_i + \lambda_{\min}(\varepsilon_4)\left(\frac{y_l}{\|y_l\|}\right). \tag{3.9}$$

For the case of the predator it will chase the prey with smaller survival value as in Eq. (3.10).

$$x_{\text{predator}} \leftarrow x_{\text{predator}} + \lambda_{\max}(\varepsilon_5)\left(\frac{y_r}{\|y_r\|}\right) + \lambda_{\min}(\varepsilon_6)\left(\frac{x_i' - x_{\text{predator}}}{\|x_i' - x_{\text{predator}}\|}\right), \tag{3.10}$$

where y_r is a random vector for the randomness movement of the predator, ε_i is a random number between 0 and 1 from a uniform probability distribution and x_i' is a prey with least survival value. If $x_i' = x_{\text{predator}}$, we ignore the second expression in Eq. (3.10), that implies the predator moves randomly.

In each iteration of the algorithm, the prey and predator set is updated. The algorithm steps can be summarized as:

(1) Set algorithm parameters and generate random feasible solutions.
(2) Calculate survival value for each solution and identify preys, predator and the best prey.
(3) Update the solutions by moving according to the formula given in Eqs. (3.8)–(3.10).
(4) If termination criteria is met terminate, else go back to Step 2.

The pseudo code of the algorithm is given in Tables 1–3.

Table 1. Pseudo code for PPA.

```
Algorithm 1: PPA

$\lambda_{\max}$   //Algorithm parameter
$f(x)$   //Objective function
Generate a set of initial solutions; $\bar{X} = \{\bar{x}_i \mid i = 1, 2, \ldots, m\}$
for index = 1 to MaxGen
  Calculate survival value for each solution member, SV($\bar{x}_i$)
  $x_{\text{predator}} = \bar{x}_j$ such that $\text{SV}(\bar{x}_j) \le \text{SV}(\bar{x}_i), \forall \bar{x}_i \in \bar{X}$
  Set of Prey $= \bar{X} \setminus x_{\text{predator}} = X = \{x_i \mid i = 1, 2, \ldots, m-1\}$
  for $i = 1$ to $m-1$
    MovePrey($x_i, x_{\text{predator}}, X, \text{SV}(x_j)\ \forall x_j \in X$)
  end
  MovePredator($x_{\text{predator}}, X, \text{SV}(x_j)\ \forall x_j \in X$)
  $\bar{X} = X \cup x_{\text{predator}}$   //Update solution set
end
```

Table 2. Pseudo code to move the predator.

```
Algorithm 2: MovePredator($x_{\text{predator}}, X, \text{SV}(x_j)\ \forall x_j \in X$)

$\lambda_{\max}, \lambda_{\min}$   //Algorithm parameter
$x'_{\text{prey}} \in X$, where $\text{SV}(x'_{\text{prey}}) \le \text{SV}(x_j)\ \forall x_j \in X$
$y_r$   //Random direction
Update $x_{\text{predator}}$ using Eq. (3.9)
Return $x_{\text{predator}}$
```

Table 3. Pseudo code for moving preys.

```
Algorithm 3: MovePrey($x_i, x_{\text{predator}}, X, \text{SV}(x_j)\ \forall\, x_j \in X$)

$\lambda_{\min}, \lambda_{\max}, p_{\text{follow-up}}\eta, \beta, \omega$   //Algorithm parameter
$A = \{x_j \in X \,|\, \text{SV}(x_j) > SV(x_i)\}$
If ($A = \{\}$)
  Generate random $k$ unit directions, $u_1, u_2, \ldots, u_k$
  $z_j = x_i + (\lambda_{\min})(\varepsilon_1)(u_j)$   //$\varepsilon_1$ is a random number between 0 and 1
  $z_{k+1} = x_i$   //$u_{k+1} = 0$
  $\text{SV}(z_j), j = 1, 2, \ldots, k+1$   //Compute survival value
  $x_i = z_l$, such that $\text{SV}(z_l) \geq \text{SV}(z_j)\ \forall\, j$
else
  if (rand $\leq p_{\text{follow-up}}$)
    Calculate $y_i$ using Eq. (3.3)
    Generate random $k$ unit direction, $y_1, y_2, \ldots, y_k$
    $z_j = x_i + \lambda_{\min} y_j$
    $\text{SV}(z_j), j = 1, 2, \ldots, k$   //Compute survival value
    $y_r = y_j$, such that $\text{SV}(x_i + \lambda_{\min} y_j) \geq \text{SV}(z_l)\ \forall\, l$
    Update $x_i$ using Eq. (3.8)
  else
    Generate a random unit direction $y_r$
    Calculate $d_1$ and $d_2$ using Eqs. (3.4) and (3.5)
    if ($d_1 \leq d_2$)
      $y_r = -y_r$
    end
    $x_i = x_i + (\lambda_{\max})(\varepsilon_4)(y_r)$, where $\varepsilon_4$ is a random number
  end
end
Return $x_i$
```

Even though PPA is a new metaheuristic algorithm, there are other algorithms which somehow include predator and prey in their names. Haynes and Sen[30] discussed on evolving behavioral strategies in predator and preys based on the predator prey pursuit. The paper considers how to surround a prey in order to capture it in a grid space. This is completely different from our proposed algorithm on the fact that PPA does not consider a grid space; furthermore the updating algorithm operator and how the predator hunts are completely different. The other algorithm is Spiral predator prey approach for multiobjective optimization which was proposed by Laumanns *et al.*[31] This algorithm is basically designed for multiobjective optimization in which it uses the concept of graph theory where each prey will be placed in the vertices of the graph. It uses $(1 + n)$-evolutionary strategy. Hence unlike PPA, the number of solutions can grow through the iterations. The operators involved in the two algorithms to update the solutions are quite different as there is no dying but replacing a solution exists in PPA. PPA also uses an operator to follow better solutions which is not the case in the Spiral predator prey approach. Unlike spiral predator prey approach, PPA uses the predator as an agent of exploration in the solution space.

4. A Short Comparison of PPA with GA and PSO

4.1. *PPA versus GA*

GA is a metaheuristic algorithm which imitates Darwin's theory of natural selection and survival of the fittest, as proposed by Holland in 1975. It is an evolutionary algorithm in which solutions members are encoded as chromosomes using 0's and 1's. The algorithm involves mutation and crossover operators. In the algorithm, high probability will be given for the fittest to perform crossover and mutation. Once new solutions are generated the fittest member will pass on to the next iteration.[5,6] A GA will have the following steps:

(1) Generate random set of solutions.
(2) Evaluate the fitness and choose parents by giving high probability to the fittest members.
(3) Perform crossover and mutation.
(4) Update the solution population.
(5) If a termination criterion is fulfilled stop, else go to Step 2.

GA is a population-based optimization algorithm like PPA. The basic differences between the two algorithms can be summarized as:

- In GA, unlike PPA, the new generation is not affected by the member with the worst performance.
- The operators in GA are very different from the distance operators in PPA.
- There is no need of coding and decoding as 0's and 1's in PPA.
- In PPA a solution member follows the best solutions while in GA there is no concept of following but there is crossover which gives a new solution with mixed chromosome of solutions.
- PPA is inspired by prey and predator of animals whereas GA is inspired by Darwin's theory of survival of the fittest.

4.2. *PPA versus PSO*

PSO is an optimization algorithm proposed by Kennedy and Everhart in 1995. It is inspired by the social behavior of flocks of birds. In the algorithm all solution members, so-called particles, flies through the problem space by following the particle with better performance and by tracking their best performance in previous iterations.[9] Each particle will have position, x_i, and velocity vectors, v_i, which will be updated with each iteration. The updating is done using the following two equations:

$$x_i \leftarrow x_i + v_i, \tag{4.1}$$

$$v_i \leftarrow wv_i + c_1 r_{1i}(xbest_i - x_i) + c_2 r_{2i}(gbest - x_i), \tag{4.2}$$

where r_{1i} and r_{2i} are random numbers between 0 and 1, c_1 and c_2 are called cognitive and social parameters and are positive constants with $c_1 + c_2 \leq 4$,[32] w is the inertia weight (we consider constant inertia weight), *xbest* is the best position of x_i and *gbest* is the global best.

PPA and PSO are both metaheuristic optimization algorithms which imitate some real aspect of nature. The two algorithms have the following basic differences:

- In PSO the direction is updated depending on two particles, *xbesti* and *gbest*, whereas PPA updating depends on the better solutions and is also affected by a member with worse performance.
- Unlike PPA, in PSO the distance between the best and a particle does not have an impact in the updating process.
- PSA performs the update using a memory of best solutions, *xbest* and *gbest*, but updating in PPA does not involve any memory of the previous iterations and it solely depends on the current performance.
- The search ideas behind both algorithms are different. PSO is inspired by the social behavior of birds and PPA is based on the prey and predator of animals.

5. Simulation Results

5.1. *Testing the algorithm*

To test the algorithm five test functions are used, as listed in Table 4. The follow-up probability was taken to be 0.8, τ to be one third and iteration number was taken to be 50. The value for the parameter varies on the size of the sample space. The number of initial solutions was taken to be 20. As in Eq. (2.1) we consider a maximization problem.

The results for each of the test functions are presented below:

The first test function, f_1, is the negative of Easom's function.[33] The feasible region is restricted to be between -10 and 10 for both variables. The optimum value of Easom function, f_1^*, is 1 at (π, π). It is a unimodal continuous test function. The graph and the contour graph of the function are given in Fig. 1.

Table 4. The five test problems with their corresponding feasible regions.

Function	Objective function	Feasible region
$f_1(x_1, x_2)$	$\cos x_1 \cos x_2 \, e^{-(x_1-\pi)^2-(x_2-\pi)^2}$	$-10 \le x_1, x_2 \le 10$
$f_2(x_1, x_2)$	$5e^{-(x_1-\pi)^2-(x_2-\pi)^2} + \sum_{j=1}^{2}\sum_{i=1}^{2} \varepsilon_{ij} e^{-(x_1-i)^2-(x_2-j)^2}$	$0 \le x_1, x_2 \le 10$
	ε_{ij} is a random number between 0 and 1	
$f_3(x_1, x_2)$	$\sum_{i=1}^{2} \operatorname{int}(x_i)$	$-5.12 \le x_1, x_2 \le 5.12$
$f_4(x_1, x_2)$	$-\left(\sum_{i=1}^{5} i\cos((i+1)x_1+1)\right)\left(\sum_{i=1}^{5} i\cos((i+1)x_2+1)\right)$	$-10 \le x_1, x_2 \le 10$
$f_5(x_1, x_2)$	$\sum_{i=1}^{2} \sin(x_i)\sin^{2m}\left(\frac{ix_i^2}{\pi}\right), \quad m = 10$	$0 \le x_1, x_2 \le \pi$

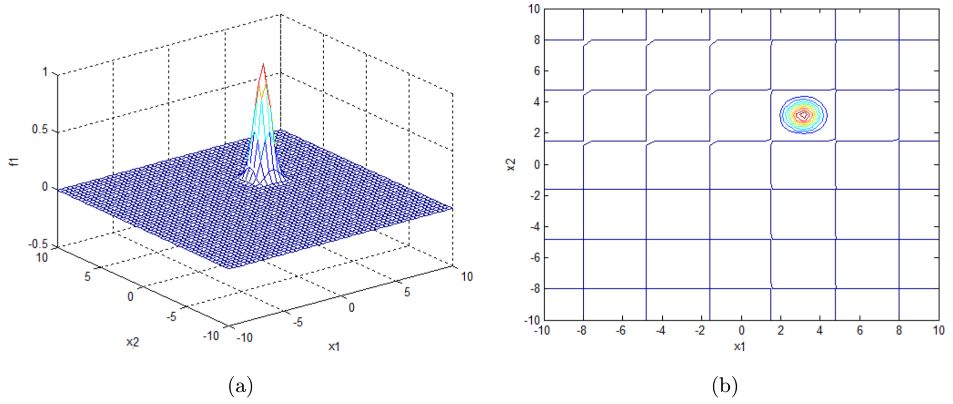


Fig. 1. (a) 3D graph of the first test function and (b) contour graph of the first test function.

The parameter $\lambda_{\max}$ was taken to be five and the result has been recorded as below, with $f_1^* = 1$ at (3.1420, 3.1412) (see Fig. 2).

In order to assess the effectiveness of PPA in achieving the optimal solution hypothesis testing is conducted. 100 trials were run with 50 iterations each; hence we will use the normal distribution. Accordingly the mean and sample standard deviation were found to be 0.99996 and 0.000283, respectively. Let the null hypothesis be, at 0.01 level of significance, the average effectiveness of PPA on this particular test problem is to achieve at least 0.99.

$$\begin{aligned} H_0 &: \bar{f} \geq 0.99 \\ H_A &: \bar{f} < 0.99 \end{aligned},$$

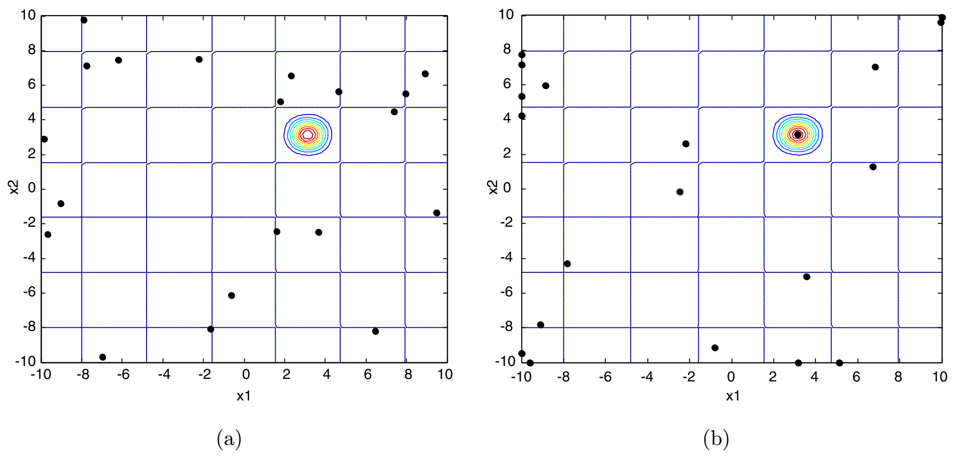
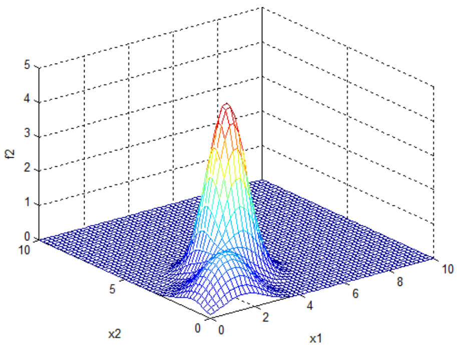
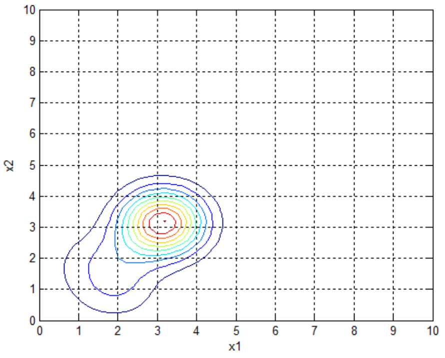


Fig. 2. (a) Contour graph of the first test function with randomly generated solutions, the black dots represent the location of the initial solutions. (b) Contour graph of the first test function after the algorithm runs for 50 iterations.



(a)



(b)

Fig. 3. (a) 3D graph of the second test function and (b) contour graph of the second test function.

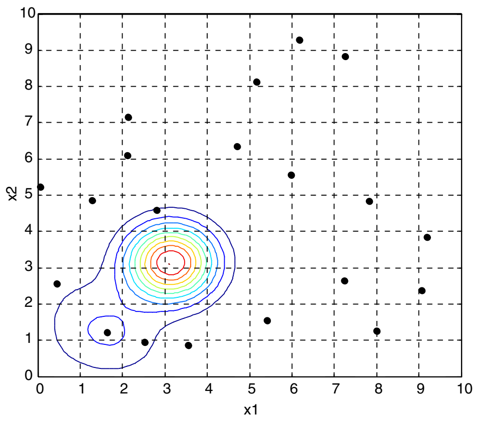
where $\bar{f}$ is the sample mean. First we need to calculate the observed z value as follows:

$$z = \frac{0.99996 - 0.99}{0.000283/\sqrt{100}} = 351.9435.$$

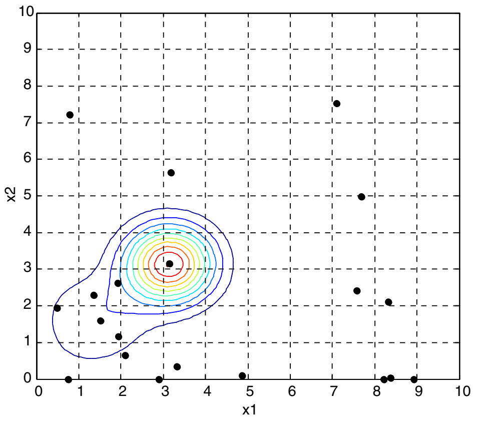
Since $z_{0.01} = 2.33 < 351.9435$; we accept the null hypothesis.

In general consider a null hypothesis of $H_0 : \bar{f} \geq 1 - t$, for some non-negative t. Suppose this hypothesis is accepted at 0.01 level of significance. This implies that the normalized observed mean should be at least 2.33.

$$\frac{0.99996 - (1 - t)}{0.000283/\sqrt{100}} = 2.33$$
$$\Rightarrow t = 0.000106.$$



(a)



(b)

Fig. 4. (a) Contour graph of the second test function with randomly generated solutions, the black dots represent the location of the initial solutions. (b) Contour graph of the second test function after the algorithm runs for 50 iterations.

Hence a null hypothesis of $H_0 : \bar{f} \geq 0.999894$ is acceptable at 0.01 level of significance.

The second test function, f_2, is a stochastic unimodal function. It has a random noise parameter. Because of the noise it is unlikely for the function to get the same value on the same point. It is taken from Ref. 34 by fixing the parameters. Its maximum value is found at (π, π). Since it is a stochastic function the maximum value depends on the random parameter. In our case f_2^* can be found to be between 5 and 5.0794 (see Fig. 3).

After running the Matlab code by setting $\lambda_{\max} = 3$, the maximum result from the 100 trials is found to be 5.0734 at (3.1378, 3.1287) (see Fig. 4).

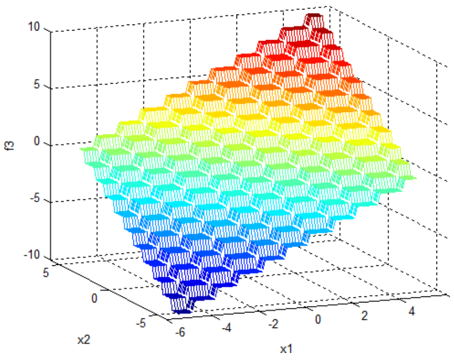
From 100 trials it is found that the mean optimum value and the standard deviation are 5.0337 and 0.0189, respectively. As done for the first test problem a null hypothesis of $H_0 : \bar{f} \geq 5.0734 - t$ is not rejected for $t \geq 0.0441$, at level of significance of 0.01. This implies that the hypothesis $H_0 : \bar{f} \geq 5.0293$ is acceptable at the given level of significance.

The third function, f_3, is a step function which is discontinuous. It has steps of flat surfaces. It is clear that flat surfaces are a challenge for optimization problem, because unless appropriate step length is specified, it is possible that the solutions will be stuck in the flat surface.[35] The maximum value, f_3^*, is 10 and it is found at (x_1^*, x_2^*) for $4.5 \leq x_1^*, x_2^* \leq 5.12$, see Fig. 5.

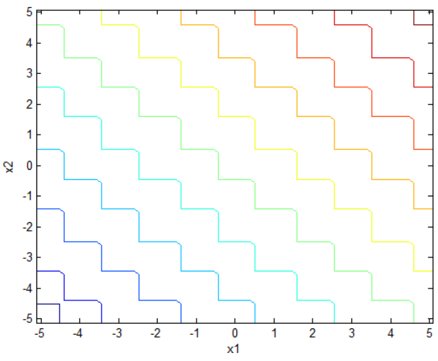
The simulation result with $\lambda_{\max} = 3$ gives $f_3^* = 10$ at (5.0841, 5.1011). For the third test problem a simulation trial was conducted 100 times; and its mean optimum value is 10 with zero deviation. Hence, PPA is efficient in solving this problem (see Fig. 6).

Our fourth test function is called the Shubert's function. It is a multimodal function with optimum value of $f_4^* = 186.7309$ at 18 points and 760 local optima.[36] The graph of the function and its counter can be seen in Fig. 7.

Here $\lambda_{\max}$ was set to be five and the simulation result $f_4^* = 186.7227$ at $(-5.8598, 6.0828)$ (see Fig. 8).



(a)



(b)

Fig. 5. (a) 3D graph of the third test function. (b) Contour graph of the third test function.

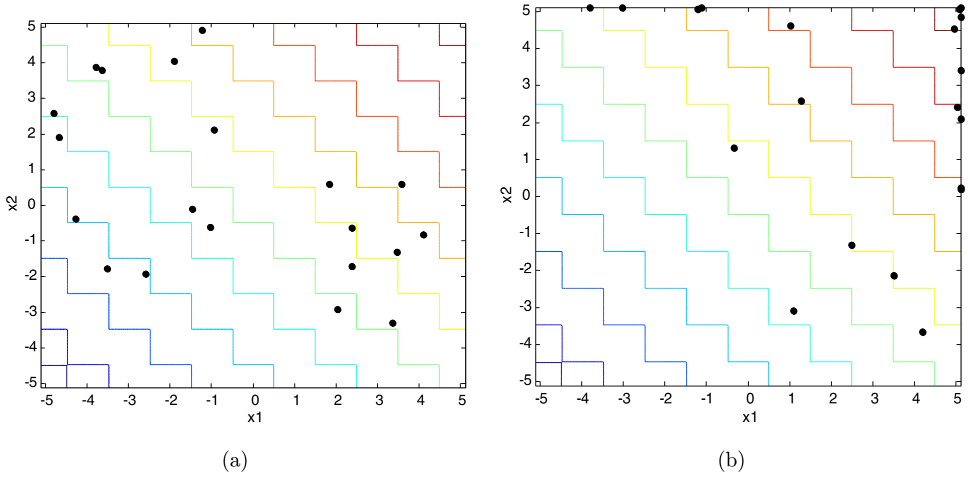


Fig. 6. (a) Contour graph of the third test function with randomly generated solutions, the black dots represent the location of the initial solutions. (b) Contour graph of the third test function after the algorithm runs for 50 iterations.

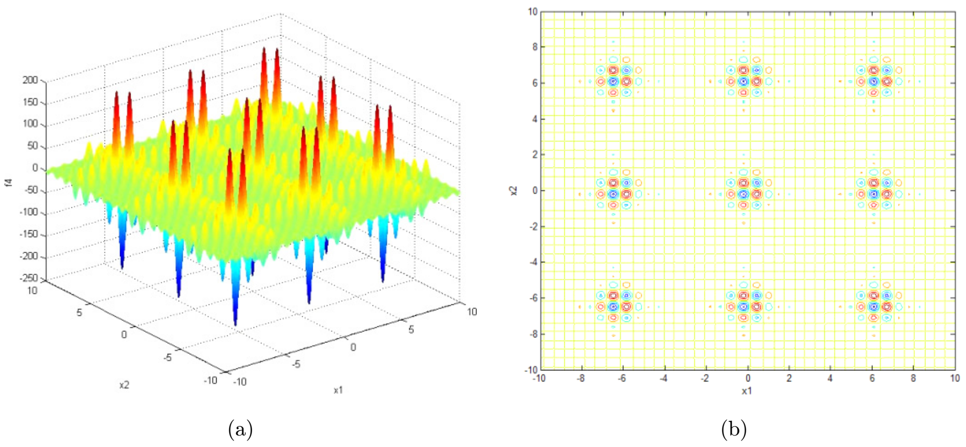


Fig. 7. (a) 3D graph of the fourth test function. (b) Contour graph of the fourth test function.

From 100 trials the mean and standard deviation of the optimum value for the fourth test problem are computed as 185.8897 and 6.658, respectively. Hence as done in the other test problems, a null hypothesis of $H_0 : \bar{f} \geq 186.7309 - t$ is acceptable at level of significance of 0.01 for $t \geq 2.3925$. In conclusion, for 0.01 level of significance $H_0 : \bar{f} \geq 184.3384$ is accepted.

The last test function we used is called the Michalewicz function.[37] It is a multimodal function attaining its maximum, $f_5^* = 1.8013$ at (2.2024, 1.5706) (see Fig. 9).

After setting $\lambda_{\max} = 1.5$ and running the Matlab code we have $f_5^* = 1.8012$ at (2.2037, 1.5093) (see Fig. 10).

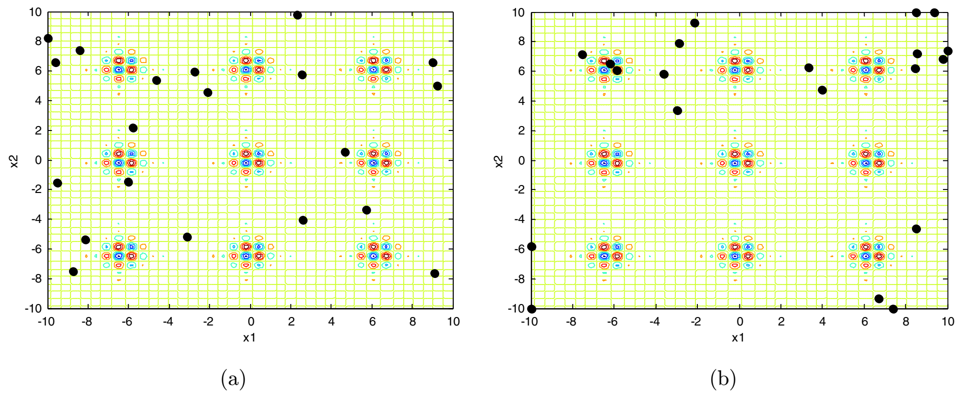


Fig. 8. (a) Contour graph of the fourth test function with randomly generated solutions, the black dots represent the location of the initial solutions. (b) Contour graph of the fourth test function after the algorithm runs for 50 iterations.

As done in the other bench mark problems the algorithm runs 100 times with 50 iterations. The sample mean and standard deviation are found to be 1.8006 and 0.0032, respectively. With 0.01 level of significance, the null hypothesis $H_0 : \bar{f} \geq 1.8013 - t$ is accepted for $t \geq 0.0014$; hence $H_0 : \bar{f} \geq 1.7999$ is acceptable at the given level of significance.

The simulation result for an arbitrary run of the code is summarized in Table 5.

As shown from the simulation results, the best prey will do a local search which helps the algorithm to do exploitation while the other preys and the predator run all over the solution space doing exploration, which minimizes the probability of getting stuck in a local optimum. If a new better solution is found the exploitation will be shifted to the place where the new best solution is found. After running the algorithm with suitable number of iterations the best prey will find the best place or the optimum while the others are scattered while exploring the solution space.

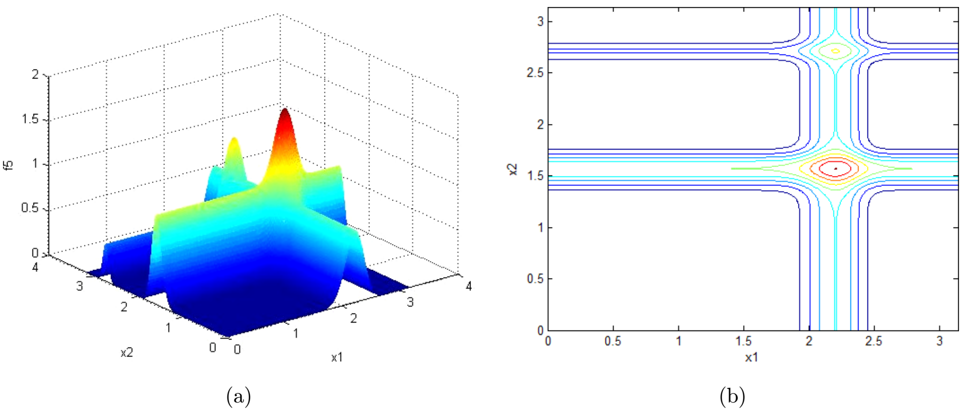


Fig. 9. (a) 3D graph of the 5th test function. (b) Contour graph of the 5th test function.

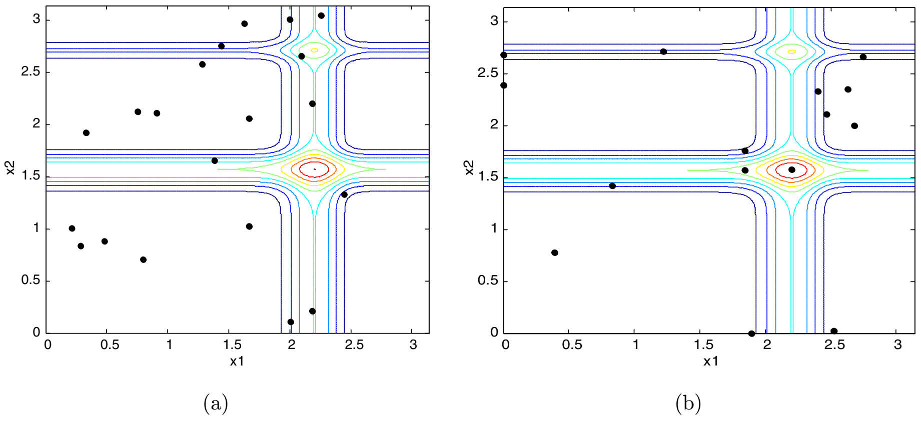


Fig. 10. (a) Contour graph of the fifth test function with randomly generated solutions, the black dots represent the location of the initial solutions. (b) Contour graph of the fifth test function after the algorithm runs for 50 iterations.

Table 5. Summary of simulation result of PPA on the five test functions, with 50 iterations.

Function	f^*	x^*	x^{*a}_{predator}	x^{*b}_{PPA}	f^{*c}_{predator}	f^{*d}_{PPA}
f_1	1	(π, π)	(4.8276, 2.7322)	(3.1420, 3.1412)	−0.0052	1
f_2	[5, 5.0794]	(π, π)	(10, 9.3590)	(3.1378, 3.1287)	0	5.0734
f_3	10	$4.5 \leq x_1^*,\ x_2^* \leq 5.12$	(−1.4876, −3.9805)	(5.0841,5.1011)	−5	10
f_4	186.7309	18 different pts[36]	(1.0613, 6.1437)	(−5.8598, 6.0828)	−81.2474	186.7227
f_5	1.8013	(2.2024,1.5706)	(3.1416, 0)	(2.2037, 1.5697)	0	1.8012

[a]x^*_{predator} is the predator at the final iteration.
[b]x^*_{PPA} is the best prey at the final iteration.
[c]$f^*_{\text{predator}} = f(x^*_{\text{predator}})$.
[d]$f^*_{\text{PPA}} = f(x^*_{\text{PPA}})$.

5.2. *Comparing PPA with GA and PSO*

We compare PPA with PSO and GA using the five test problems stated in Sec. 5.1. The same randomly generated initial population set was taken for all the algorithms and the numbers of iterations were taken to be 25, 50 and 100 for the three algorithms. The comparison was done 100 times.

In GA, the probability of crossover and mutation were taken as 0.85 and 0.3, respectively. Furthermore each variable was encoded using 20 bits of 0's and 1's. For PSO the inertia weight, w, was taken to be one. Furthermore, the cognitive and social parameters were set as $c_1 = 1.5$ and $c_2 = 2.5$. The parameters for PPA were taken as used in the previous section.

In each of the 100 runs or comparisons of the three algorithms the result was recorded and put in percentage accuracy as shown in Table 6. In the table, the second row shows the number of iterations, 25, 50 or 100, in which the comparison is done. The second column shows the interval $[f_i^* - e_j, f_i^*]$, for $j = 1, 2, 3$ and $i = 1, 2, 3, 4, 5,$

Table 6. Comparison between GA, PSO and PPA using different number of iterations and levels of accuracy.

Test function and the interval		GA			PSO			PPA		
		25	50	100	25	50	100	25	50	100
f_1	$f_1^* = 1$	6%	42%	42%	1%	9%	31%	64%	99%	100%
	[0.9980,1]	37%	68%	67%	16%	49%	98%	95%	100%	100%
	[0.9961,1]	48%	73%	75%	28%	67%	100%	97%	100%	100%
	[0.5088,1]	81%	83%	83%	100%	100%	100%	98%	100%	100%
	e^*	0.9999	0.9999	1	0.2102	0.0126	0.0029	0.9999	0.0001	0
f_2	$f_2^* \geq 5$	74%	91%	94%	58%	78%	79%	99%	100%	100%
	$f_2^* \geq 4.9$	82%	91%	94%	91%	99%	100%	100%	100%	100%
	$f_2^* \geq 4.0$	87%	91%	94%	100%	100%	100%	100%	100%	100%
	$f_2{}^* \geq 2.5$	97%	98%	99%	100%	100%	100%	100%	100%	100%
	e'^{*}[a]	3.9935	3.6020	2.3698	0.2558	0.1079	0.0766	0.0001	0	0
f_3	$F_3^* = 10$	99%	100%	100%	100%	100%	100%	100%	100%	100%
	[9, 10]	100%	100%	100%	100%	100%	100%	100%	100%	100%
	[8, 10]	100%	100%	100%	100%	100%	100%	100%	100%	100%
	e^*	1	0	0	0	0	0	0	0	0
f_4	$f_4^* = 186.7309$	0%	0%	1%	0%	0%	2%	0%	13%	36%
	$[186.72, f_4^*]$	3%	11%	28%	0%	2%	11%	12%	52%	87%
	$[186, f_4^*]$	24%	88%	100%	11%	46%	68%	76%	99%	100%
	$[185, f_4{}^*]$	47%	94%	100%	21%	62%	99%	81%	99%	100%
	e^*	70.082	14.712	0.4848	152.49	33.035	2.4723	68.497	63.164	0.031
f_5	$f_5^* = 1.8013$	23%	55%	55%	0%	3%	4%	12%	39%	75%
	$[1.8, f_5^*]$	74%	77%	77%	1%	21%	37%	74%	98%	100%
	$[1.77, f_5^*]$	88%	91%	91%	49%	68%	80%	100%	100%	100%
	$[1.7, f_5^*]$	92%	91%	91%	68%	72%	80%	100%	100%	100%
	e^*	0.6533	0.6533	0.6232	1.001	1	1	0.0051	0.0026	0.0007

[a] e'^* is the difference between 5 and the least value.

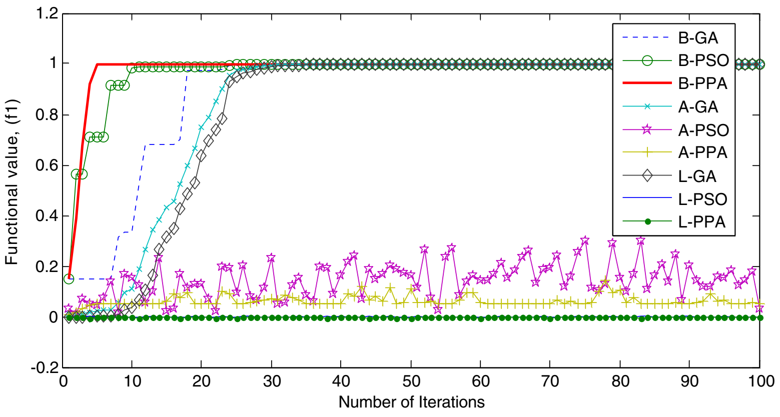
and the results from the three algorithm are counted in the interval and the percentage is put accordingly. Furthermore, the minimum e^* in which 100% of the data lies in the interval satisfying $[f_i^* - e^*, f_i^*]$ is also given for the 100 runs in each iteration number for the corresponding algorithm.

Now let us have a look at the performance of the three algorithms in terms of the best average and worst performance from a close look of a run of the algorithms. The number of iteration was taken to be 100 and from 100 trials the best, average and worst performances are recorded as shown in Table 7. Furthermore for a trial with iteration number of 100 is illustrated in Fig. 11 for each test functions.

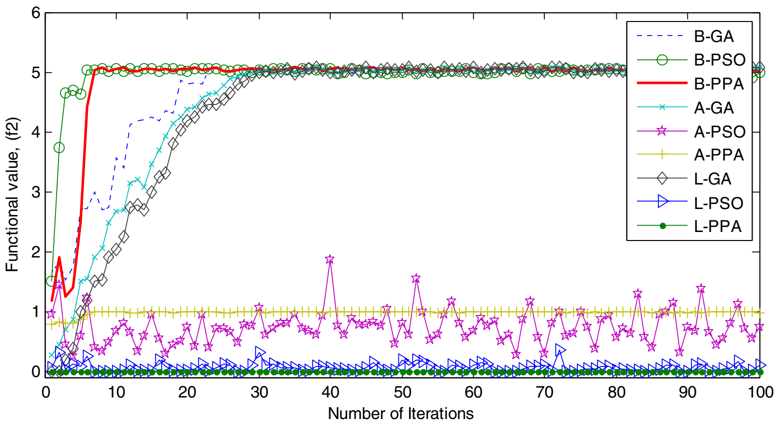
From the simulation results it is clear to see that GA does better as a group. This means in the case of GA all the solution members tend to converge to the best. Whereas in PSO and especially in PPA most of the solution members run all over the solution space to do the exploration while the best solutions keep on doing a local search or exploitation, as shown in Table 7 and Fig. 11. In the optimal solution, PPA

Table 7. Comparison of GA, PSO and PPA on the best average and worse performance.

Algorithm and performance level		f_1	f_2	f_3	f_4	f_5
	Best	**1**	**5.0192**	**10**	**186.6559**	**1.8009**
GA	Average	1	5.0192	10	186.6556	1.8009
	Worst	1	5.0192	10	186.6404	1.8009
	Best	**0.9998**	**5.0139**	**10**	**186.7255**	**1.8009**
PSO	Average	0.1630	1.5235	10	3.6085	0.3545
	Worst	−0.0048	0	10	−25.4996	0.1092
	Best	**1**	**5.0561**	**10**	**186.7304**	**1.8013**
PPA	Average	0.2310	1.2702	4.825	12.9451	0.2145
	worst	−0.0001	0	−8.95	−32.1884	0

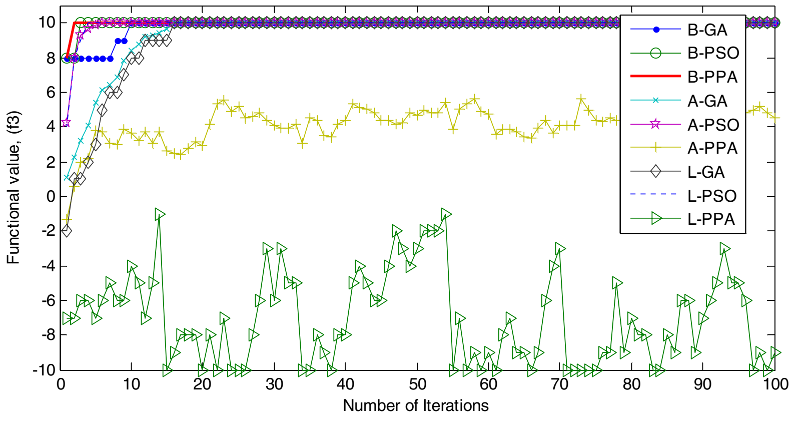


(a)

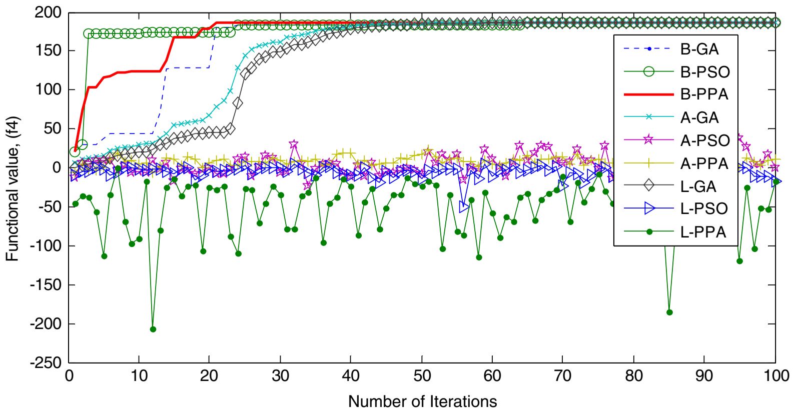


(b)

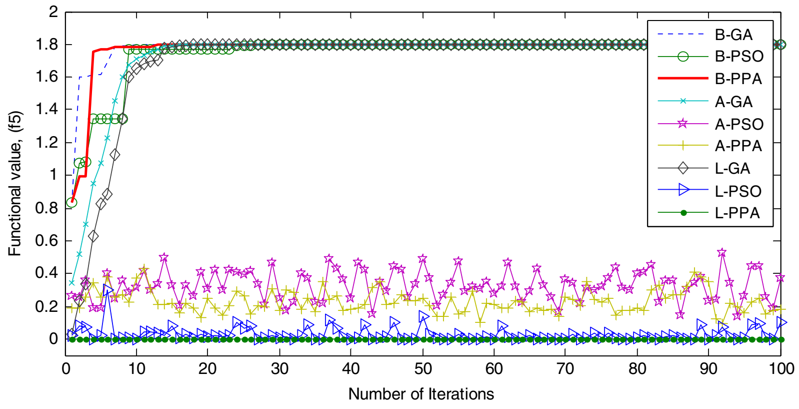
Fig. 11. Comparison of GA, PSO and PPA, each algorithm runs for 100 iterations with the same initial solution set, in the key B stands for best, A for Average and L for least. The best, average and least performance of the three algorithms (a) on the first test function, (b) on the second test function, (c) on the third test function, (d) on the fourth test function and (e) on the fifth test function.



(c)



(d)



(e)

Fig. 11. (*Continued*)

does better in all the test functions and in large number of iterations for test function 5. Generally, it is shown that PPA attains the optimum solution better than GA and PSO in the given test functions, number of iterations and given set of parameters.

6. Conclusion

In this paper, we introduce a new bio-inspired metaheuristic algorithm for optimization problems. The algorithm imitates the prey and predator interaction, where the predator runs after the prey. The algorithm works by assigning a fitness for each candidate solution called the survival value depending on its performance. A member with the best surviving value will be assigned to be the best prey and focuses in the exploitation task whereas the other members will be assigned as preys, except the member with least survival value which is the predator, to do exploitation and exploration with more focus on the exploration. The predator will chase after the prey with the least survival value.

It is shown that the algorithm easily finds the best solution as compared to GA and PSO in the selected five test problems. The comparison on the average performance and the least performance shows that PPA focuses on exploring while keeping the best to do exploitation only. If a better solution is found, PPA will keep the current best. Future works are expected to be done on the parametric control, complexity analysis, convergence analysis, testing the algorithm in solving real life problems and extending it for the multiobjective optimization problem case.

Acknowledgments

This work was supported in part by Universiti Sains Malaysia (USM) Malaysia Fundamental Research Grant Scheme (FRGS) No. 203/PMATHS/6711319.

References

1. T. K. Papathanassiou, S. P. Filopoulos and G. J. Tsamasphyros, Optimization of composite patch repair processes with the use of genetic algorithms, *Optimization and Engineering* **12** (2011) 73–82, doi:10.1007/s11081-010-9116-0.
2. H. Tang, L. Tian and L. Jia, Inventory-transportation integrated optimization problem: A model of product oil logistics, *International Journal of Nonlinear Science* **8** (2009) 92–96.
3. N. Svilen and Z. Tsvetan, Load optimization in training the olympic rowing (Skiff) champion from Beijing, 2008: Case study, *Serbian Journal of Sports Sciences* **5** (2011) 41–50.
4. E.-G. Talbi, *Metaheuristics: From Design to Implementation* (John Wiley and Sons, Inc., New Jersey, 2009).
5. M. Negnevitsky, *Artificial Intelligence: A Guide to Intelligent Systems*, 2nd edn. (Pearson Education Limited, Edinburgh Gate, UK, 2005).
6. X.-S. Yang, *Nature-Inspired Metaheuristics Algorithm*, 2nd edn. (Luniver press, Frome, UK, 2010).

7. S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by simulated annealing, *Science* **220** (1983) 671–680.
8. M. Dorigo and C. Blumb, Ant colony optimization theory: A survey, *Theoretical Computer Science* **344** (2005) 243–278.
9. K. Kameyama, Particle swarm optimization-A survey, *IEICE Transactions on Information and Systems* **E92-D**(7) (2009) 1354–1361.
10. O. Montiel, O. Castillo, P. Melin, A. R. Díaz and R. Sepúlveda, Human evolutionary model: A new approach to optimization, *Information Sciences* **177**(10) (2007) 2075–2098.
11. F. Valdez, P. Melin and O. Castillo, An improved evolutionary method with fuzzy logic for combining particle swarm optimization and genetic algorithms, *Applied Soft Computing* **11**(2) (2011) 2625–2632.
12. L. Nie, X. Xu and D. Zhan, Collaborative planning in supply chains by Lagrangian Relaxation and genetic algorithm, *International Journal of Information Technology & Decision Making* **7**(1) (2008) 183–197.
13. C.-F. Juang, A hybrid of genetic algorithm and particle swarm optimization for recurrent network design, *IEEE Transactions on Systems, Man, and Cybernetics, Part B* **34**(2) (2004) 997–1006.
14. R. Thamilselvan and P. Balasubramanie, Integrating genetic algorithm, tabu search approach for job shop scheduling, *International Journal of Computer Science and Information Security* **2**(1) (2009) 1–6.
15. S. L. Tilahun and H. C. Ong, Vector optimisation using fuzzy preference in evolutionary strategy based firefly algorithm, *International Journal of Operational Research* **16**(1) (2013) 81–95.
16. H.-Y. Lee, A decision support system for exposition timetabling using ant colony optimization, *International Journal of Information Technology & Decision Making* **11**(3) (2012) 609–626.
17. E. Hopper and B. C. H. Turton, A review of the application of meta-heuristic algorithms to 2D strip packing problems, *Artificial Intelligence Review* **16** (2001) 257–300.
18. H. H. E. Morra, A. U. Sheikh and A. Zerguine, Application of heuristic algorithms for multiuser detection, *IEEE ICC 2006 Proc.* (2006), pp. 3127–3161.
19. L. Mendes, P. Godinho and J. Dias, A Forex trading system based on a genetic algorithm, *Journal of Heuristics* **18**(4) (2012) 627–656.
20. S. L. Tilahun and H. C. Ong, Bus timetabling as a fuzzy multiobjective optimization problem using preference based genetic algorithm, *PROMET — Traffic & Transportation* **24**(3) (2012) 183–191.
21. C. J. Krebs, *Ecology*, 6th edn. (Pearson Education, Inc., San Francisco, 2009).
22. D. T. Krohne, *General Ecology*, 2nd edn. (Brooks/Cole, Pacific Grove, USA, 2001).
23. R. A. Danao, *Mathematical Methods in Economics and Business* (University of the Philippines press, Quezon city, Philippines, 2007).
24. T. M. Smith, *Elements of Ecology*, 6th edn. (Pearson Education, Inc., San Francisco, 2009).
25. D. Estigarribia, *Cheetahs* (Marshall Cavendish Corporation, China, 2005).
26. G. B. Schaller, *The Serengeti Lion: A Study of Predator-Prey Relations* (The University of Chicago press, Chicago, 1972).
27. Marshall Cavendish Corporation, Endangered Wildlife and Plants of the World (Marshall Cavendish Corporation, Tarrytown, NY, 2001).
28. H. V. Lawick, *Savage Paradise: The Predators of Serengeti* (W. Morrow, New York, 1977).
29. D. C. Heard, The effect of wolf predation and snow cover on musk-ox group size, *American Naturalist* **139** (1992) 190–205.

30. T. Haynes and S. Sen, Evolving behavioral strategies in predators and prey, in *Adaptation and Learning in Multiagent Systems*, LNAI (Springer Verlag 1996), 113–126.
31. M. Laumanns, G. Rudolph and H.-P. Schwefel, A spatial predator-prey approach to multi-objective optimization: A preliminary study, Lecture Notes in Computer Science, Vol. 1498 (Springer Berlin Heidelberg, 1998), pp. 241–249.
32. J. Kennedy, The behavior of particles, in *Proc. 7th Int. Conf. Evolutionary Programming*, San Diego, USA (1998), pp. 581–589.
33. E. J. S. Pires, J. A. T. Machado, P. B. de M. Oliveira, J. B. Cunha and L. Mendes, Particle swarm optimization with fractional-order velocity, *Nonlinear Dynamics* **61** (2010) 295–301.
34. X.-S. Yang, Firefly algorithm, stochastic test functions and design optimisation, *International Journal of Bio-Inspired Computation* **2** (2010) 78–84.
35. D. Karaboga and S. Okdem, A simple and global optimization algorithm for engineering problems: Differential evlolution algorithm, *Turkish Journal of Electrical Engineering* **12** (2004) 53–60.
36. L. Idoumghar, M. Melkemi and R. Schott, A novel hybrid evolutionary algorithm for multi-modal function optimization and engineering applications, in *Proc. 13th IASTED Int. Conf. Artificial Intelligence and Soft Computing — ASC 2009*, Spain, Palma de Mallorca (ACTA press, 2009), pp. 87–93.
37. C.-F. Huang and L. M. Rocha, Exploration of RNA editing and design of robust genetic algorithms, in *Proc. 2003 IEEE Congress on Evolutionary Computation*, Australia, Canberra (IEEE Press, 2003), pp. 2799–2806.