SCIENCE CHINA
Information Sciences

• **RESEARCH PAPER** •

October 2012 Vol. 55 No. 10: 2369–2389
doi: 10.1007/s11432-012-4548-0

Lion pride optimizer: An optimization algorithm inspired by lion pride behavior

WANG Bo, JIN XiaoPing & CHENG Bo*

State Key Laboratory of Automotive Safety and Energy, Tsinghua University, Beijing 100084, *China*

Received April 6, 2011; accepted June 24, 2011; published online March 12, 2012

Abstract In this paper, we report a novel optimization algorithm, lion pride optimizer (LPO), which is inspired by lion pride behavior. The framework is mainly based on lion prides' evolution process and group living theory. In a lion pride, brutal competition of individuals happens among male lions both within and among prides; on the other hand, each member plays an important role in the persistence of a lion pride. Based on this framework, concepts from lion prides behavior, e.g., the strongest males occupy nearly all mating resources, and if a new cohort of males is able to take over a pride, they will seek to kill young cubs sired by their predecessors, are employed metaphorically to design optimum searching strategies for solving continuous optimization problems. From the studies of the algorithm property, it is found that the LPO algorithm is not sensitive to most parameters, which shows the robustness of the algorithm and the parameters are not problem-dependent. Central tendency of the algorithm is not found. It is found that the pride update strategy and brutal competition of individuals are two main factors that contribute to the performance of LPO. According to the test results on 23 famous benchmark functions, the LPO algorithm has better performance than the other seven state-of-the-art algorithms on both unimodal and multimodal benchmark functions; in the test of high-dimensional multimodal problems, LPO outperforms the other five algorithms on all benchmark functions.

Keywords animal behavior, evolutionary algorithm, lion pride, optimization, swarm intelligence

Citation Wang B, Jin X P, Cheng B. Lion pride optimizer: An optimization algorithm inspired by lion pride behavior. Sci China Inf Sci, 2012, 55: 2369–2389, doi: 10.1007/s11432-012-4548-0

1 Introduction

Research on social insect behavior has provided computer scientists with powerful methods for designing distributed control and optimization algorithms. These techniques are being applied successfully to a variety of scientific and engineering problems [1]. Among them, the most successful are evolutionary algorithms (EAs) which draw inspiration from evolution by natural selection [2]. Currently, there are several different types of EAs which include genetic algorithms (GAs), genetic programming (GP), evolutionary programming (EP), and evolutionary strategies (ES) [3].

Genetic algorithm (GA) is a famous stochastic optimization algorithm pioneered by Holland [4]. It mimics the evolutionary process of a population of individuals over time. The hallmark of GA is the provision of a population (multiple parallel search capability), selection (survival of the fittest), crossover (sexual reproduction), and mutation (random incremental changes) [5]. Evolutionary strategies (ES) and

*Corresponding author (email: chengbo@tsinghua.edu.cn) evolutionary programming (EP) are closely related to GA, and the schools of GA, ES, and EP are best viewed as different dialects of the same fundamental evolutionary algorithm (EA) [6].

However, the capability of traditional EP, ES, and GA is limited. Fast evolutionary programming (FEP) and fast evolutionary strategies (FES) were proposed to enhance the convergence rates [2,7], and many improvements were conducted to enhance the performance of GA [5]. For example, many authors advocated that an adaptive mutation schedule would enhance GA performance significantly [8]. It was confirmed by numerous empirical studies that duplicate removal can enhance the performance of GA significantly [9–11]. From another point of view, premature convergence is commonly thought of as an undesirable phenomenon to cause poor performance of GA [5]. Numerous operators that modify the selection, crossover or mutation to diversify the population have been proposed. Two well-known examples are fitness sharing [12] and island model [13]. Another solution that adaptively mutates and never revisits was proposed in [5]. On the other hand, differential evolution (DE), a special case of GAs, has been proposed and was proved that it could solve a broad range of optimization problems [14,15].

In recent years, evolutionary computational methods, which belong to the swarm intelligence (SI) category, have proven to be promising tools to solve many engineering and financial problems. They have been found to be powerful methods in domains where analytic solutions have not proven effective. Currently, the most famous SI algorithms are ant colony optimization (ACO) and particle swarm optimizer (PSO), ACO was developed based on the behavior of real ant colonies, in particular by their foraging behavior [1], and it employs artificial ants that cooperate to find good solutions for discrete optimization problems. ACO was mainly applied to vehicle routing, scheduling, timetabling, traveling salesman problem, routing in packet-switched networks, and data mining [16]. PSO gleaned ideas from the social behavior of bird flocks and fish schools [17]. PSO was employed to solve many problems, such as nonlinear optimization problem and min-max optimization. For more information of recent development of PSO, readers are directed to a comprehensive survey of PSO in [18].

The two SI algorithms mentioned above were inspired by some aspects of animal behavior [2]. Actually, there were many algorithms belonging to the category of the so-called animal behavior-inspired algorithms. Area-restricted searching behavior, on which predators initially search for prey at a chosen pace and directions, has inspired synthetic predator search (SPS) algorithm for solving combinatorial optimization problems [19]. Society and civilization algorithm, inspired from the intra and intersociety interactions within a formal society and the civilization model (e.g., human and social insect societies), was designed to solve single objective constrained optimization problems [20]. Predator-prey model from animal behavior has been used to approximate the shape of the Pareto-optimal set of multi-objective optimization problems [21].

Recently, the behavior of bacteria inspired two kinds of algorithms. One is bacterial foraging optimization (BFO) designed from the natural foraging strategy of bacteria [22], and has been applied to real-world problems such as optimal controller design, harmonic estimation, transmission loss reduction, active power filter synthesis, and learning of artificial neural networks. The performances of BFO on unimodal functions and multimodal functions with a few local minima were examined in [23]. The other one is bacteria chemotaxis optimization (BCO), which is based on a bacteria chemotaxis model. However, BCO performs similarly to standard evolution strategies and worse than evolution strategies with enhanced convergence properties [24]. Thus the potential of bacteria-inspired algorithms remains to be exploited.

A recently published paper reported a novel optimization, group search optimizer (GSO), which was inspired by animal behavior, especially animal searching behavior. The framework is mainly based on the producer-scrounger model, which assumes that group members search either for "finding" (producer) or for "joining" (scrounger) opportunities. The performance of GSO proved that it outperformed several state-of-the-art algorithms [2].

In this paper, inspired by animal behavior, especially lion prides behavior, we propose a novel optimization algorithm, called lion pride optimizer (LPO), primarily for continuous optimization problems. African lions live in social groups called prides and exhibit group territorial behavior [25]. In a pride, male lions are usually transient members of prides; they are replaced by new males who become resident for 2–4 years before being replaced by yet another coalition [26]. Both sexes are polygamous and breed throughout the year, but females are usually restricted to the one or two adult males of the pride. Sexual maturity male cubs are expelled from the pride and become nomads until they are old enough to try to take over another pride. Many adult males remain nomads for life, mating opportunities for nomad males are rare, and competition between male lions to defend a pride's territory and mate with the pride females is fierce. Cooperating partnerships of 2–4 males are more successful at maintaining tenure with a pride than individuals [27]. If a new cohort of males is able to take over a pride, they will seek to kill young cubs sired by their predecessors. Thus, the members of a pride, especially male lions, are not steady. Frequent male takeovers may be an explanation of why the male lion can be very strong and become "king of beasts".

It is thought that Darwin's natural selection theory indicates the inner rule of biology evolution. Survival of the fittest (SF) is the main idea [4]. For a lion pride, only the few strongest male lions can remain in the pride. Thus, SF model is adopted in the LPO algorithm. In the SF model, three main evolution strategies are developed according to the lion pride behavior: 1) Two best members occupy all the mating resources of the pride; 2) the best member of the new offspring is trained to be stronger; 3) evolution stagnation leads to the takeover by new individuals, and long evolution stagnation leads to a mutation to the best member.

Optimization, which is a process of seeking optima in a search space, is analogous to the evolution process (the best member represents the optima obtained) of animals in nature. The LPO proposed in this paper is a population-based optimization algorithm that employs SF model as the framework. In the framework, behaviors and strategies of lion prides are employed to design the special evolution strategies. One-dimension search is adopted in the training and mutation to enhance the constringency speed, and a new crossover strategy is adopted to avoid premature convergence.

To evaluate the performance of the LPO algorithm comprehensively, a precise and long-term study on a set of 23 benchmark functions is presented in this paper. For the comparison of algorithms' capability, the results of newly published GSO, GA, PSO [2], EP [7], ES [28], and their improved algorithms FEP [7], FES [28] are employed. The 23 benchmark functions are adopted (or partly adopted) to evaluate EAs by many researches [2,5,7,23,28,29]. However, their dimensions are small (up to 30) compared with real-world optimization problems which usually involve hundreds or even thousands of variables [2]. In order to further investigate whether LPO is able to solve large-scale optimization problems well, we tested LPO algorithm on six multimodal benchmark functions (e.g., f_8–f_{13} studied in this paper) in 300 dimensions.

This paper is organized as follows. In the rest parts, we will introduce LPO and its implementation details in Section 2. The experimental results of the proposed LPO and the comparison with other algorithms are presented in Section 3. The algorithm property is studied in Section 4. As a new population-based swarm intelligence and evolution algorithm, LPO does share some characteristics with other swarm intelligence algorithms and evolution algorithms, such as PSO, GSO, ACO and GAs. Thus the discussion of the differences and similarities between LPO and GSO, ACO, PSO, GAs are presented in Section 5. The conclusions are obtained in Section 6.

2 Lion pride optimizer

2.1 Basic property

LPO is also a population-based optimization algorithm. A group of lions live together, and the members compete with each other for the king position. The kings mate with nearly all female lions in the group; they reproduce offspring and defend their territory against the invading of other groups; the offspring will grow up and become strong enough to take over the king position or be driven out from the group if they are not strong enough. The kings of the group need to become stronger to defend the menace both within and without the group; otherwise, the kings may be replaced by other male lions within or without the group. If a new cohort of males is able to take over a pride, they will seek to kill young cubs sired by their predecessors.

The population of a lion pride is set to 50, which is larger than the average group size of real lion pride. The population (population size: M) of the LPO algorithm is called pride and each individual is called member. For an n-dimensional optimization problem, the ith member in the kth generation of the pride is represented by an n-dimensional vector $X_i^k \in R^n$, $i = 1, 2, \ldots, M$, which represents or equals the real independent variable.

For minimum problems, the fitness function $f(x)$ can be set to be the real minimum function, while for maximum problems, $f(x)$ can be set to be the negative real minimum function. The optimization process is along with the generation change; therefore, the generation number of the LPO is a main factor that can control the optimization process.

2.2 Surviving of two best members

In LPO, members of a pride can be considered as either male or female if it is needed to enhance the evolution efficiency. As mentioned before, cooperating partnerships of 2–4 males are more successful at maintaining tenure with a pride than individuals [27], and the male lions, which were evicted from a pride when they become adult, may form a coalition with other males [30], and as the randomicity of the evolution process, it is also possible that the best members reproduce worse offspring in the next generation. Therefore, in the SF model, the two best members of each pride are assumed to be males that occupy all the mating resources of the pride. Then

$$X_{b1}^k \in \{X_i^k, i = 1, 2, \ldots, M\}, \text{s.t. } f(X_{b1}^k) = \max\{f(X_i^k), i = 1, 2, \ldots, M\} \tag{1}$$

is the best member in the k generation. Similarly, X_{b2}^k is the second best member.

The surviving of two best members is somewhat similar to the niche method. However, the only sameness with niche method is that two best members are saved. Any strategy in the niche method is not used in LPO. As the competition among different members is cruel and the pride is updated frequently, the surviving of just one best member is apt to lead to premature convergence; therefore, two best members are saved in LPO.

2.3 Novel crossover strategy

As mentioned before, two best members of a pride will be saved in the optimization process. If traditional crossover strategy [4] is adopted in our algorithm, it is similar to the situation that the two members are copied for many times. However, it is confirmed by numerous empirical studies that duplicate removal can enhance the performance of EAs significantly [9–11]. Thus a new mating strategy that can avoid duplication is sought and adopted in LPO.

In a pride, members are very likely to be around with the two best members. Hence, the direction that connects one member and one best member is very likely to be particular, and all these directions are likely to cover all optimization directions. As there are two best members and two directions that connect two members (e.g., for member A and member B, one direction is from A to B, and the other direction is from B to A). Hence, each female can reproduce four children with the two best members but just one in four of them can survive at last. The detailed strategy is described in the following formula, and the convergence process of LPO is sketched in Figure 1.

$$\begin{aligned}
x_{4i-3}^{k+1} &= X_{b1}^k + mc_i^k(X_{b1}^k - X_i^k),\ i = 1, 2, \ldots, M,\\
x_{4i-2}^{k+1} &= X_{b1}^k - mc_i^k(X_{b1}^k - X_i^k),\ i = 1, 2, \ldots, M,\\
x_{4i-1}^{k+1} &= X_{b2}^k + mc_i^k(X_{b2}^k - X_i^k),\ i = 1, 2, \ldots, M,\\
x_{4i}^{k+1} &= X_{b2}^k - mc_i^k(X_{b2}^k - X_i^k),\ i = 1, 2, \ldots, M,
\end{aligned} \tag{2}$$

where x_{4i}^{k+1} and x_{4i-1}^{k+1} are the children of X_{b2}^k and X_i^k, x_{4i-2}^{k+1} and x_{4i-3}^{k+1} are the children of X_{b1}^k and X_i^k, mc_i^k is the mating coefficient for X_i^k. If the mating coefficient is a constant, the offspring will be reproduced in a fixed step, the randomicity of the algorithm is reduced and the algorithm capability may

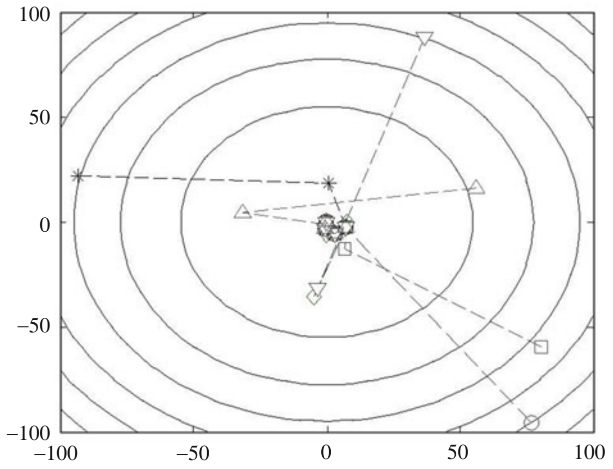

Figure 1 Sketch map of six members' convergence (minimum in [0,0]) of LPO in five iteration. Each shape is a member.

be unstable. Thus, in LPO, for a pride, there is a basic mating coefficient mc_0, and mating coefficient is calculated as follows:

$$mc_i^k = mc_0(\text{rand}(1,1) - 0.5),\ i = 1, 2, \ldots, M, \tag{3}$$

where rand(1,1) is a random number in (0,1). Then there is a different mc_i^k for X_i^k. Among the four children, just one in four of them can survive at last,

$$X_1^{k+1} = x_\alpha^{k+1}, \alpha \in \{1, 2, \ldots, 4M\}, \text{s.t. } f(x_\alpha^{k+1}) = \max\{f(x_i^{k+1}), i = 1, 2, \ldots, 4M\}, \tag{4}$$

where X_1^{k+1} is the first member in $k+1$ generation. Similarly, X_j^{k+1} is jth member $(j = 2, 3, \ldots, M)$.

2.4 Scaling of searching space

Although the pride is a stable social unit, it is a fission-fusion social group, in which individuals are typically found in a range of subgroup sizes and compositions [31]. Male coalitions challenge one another for pride residency [26], and it was indicated that male lions may use lethal aggression to tip the balance of power in favor of their prides. Numerical advantage in territorial competition is a primary benefit of group living in lions and may have been important in the evolution of lion sociality [31]. The incoming males kill or evict dependent offspring sired by the previous coalition, so as to accelerate the mothers' return to sexual receptivity [26]. Thus in LPO, if the evolution of a pride stops (called stagnation) for a certain period (bigger than a threshold *ths*), all members of the pride will be replaced by a new set of members. In order to keep the convergence speed, the two best members of the original pride will be saved and the new members will be created in a smaller space around the best members.

However, premature convergence is an undesirable phenomenon often reported in the literature that causes poor performance of EAs. Therefore, in order to avoid premature convergence, another strategy is imported, and then the search space will not decrease at all times and will return to the original search space after a long period. Then

$$L^k = \begin{cases} L^0, & \text{mod}(ts, k_2) = 0, \\ \dfrac{L^0}{k_1 \ln(ts+2) - k_1 + 1}, & \text{mod}(ts, k_2) \neq 0, \end{cases} \tag{5}$$

where $L^k \in \mathbb{R}^n$ is the length of searching space in n dimension space, L^0 is the original length, ts is the stagnation times, k_1, k_2 are constants, $\text{mod}(ts, k_2)$ is the arithmetical compliment of ts/k_2. If $\text{mod}(ts, k_2) = 0$, the center of searching space is original one. If $\text{mod}\ (ts, k_2) \neq 0$, the center is X_{b1}^k.

By formula (4), a new generation is created. For a lion pride, just the two best members of each pride are assumed to be the males that occupy all the mating resources of the pride. Thus, the two best members must be strong enough to resist the challenge from other males for pride residency. Hence in LPO, the two potentially best members are further optimized by searching in the potentially best evolution direction. The selection of potentially best evolution directions is as follows:

$$\begin{aligned} D_{b1}^{k+1} &= \frac{x_\alpha^{k+1} - X_{b1}^k}{\|x_\alpha^{k+1} - X_{b1}^k\|_2}, \text{s.t.}, \frac{f(x_\alpha^{k+1}) - f(X_{b1}^k)}{\|x_\alpha^k - X_{b1}^k\|_2} = \max\left\{\frac{f(x_i^{k+1}) - f(X_{b1}^k)}{\|x_i^{k+1} - X_{b1}^k\|_2}, i = 1, 2, \ldots, 2M\right\}, \\ D_{b2}^{k+1} &= \frac{x_\beta^{k+1} - X_{b2}^k}{\|x_\beta^{k+1} - X_{b2}^k\|_2}, \text{s.t.}, \frac{f(x_\beta^{k+1}) - f(X_{b2}^k)}{\|x_\beta^k - X_{b2}^k\|_2} = \max\left\{\frac{f(x_j^{k+1}) - f(X_{b2}^k)}{\|x_j^{k+1} - X_{b2}^k\|_2}, j = 1, 2, \ldots, 2M\right\}, \end{aligned} \tag{6}$$

where x_i^{k+1} and x_j^{k+1} is the children of X_{b1}^k and X_{b2}^k respectively, D_{b1}^{k+1} is the best evolution direction of the best member in the $k+1$ generation, and D_{b2}^{k+1} is the best evolution direction of the second best member in the $k+1$ generation. Then, according to the best evolution directions, the two best members are further optimized as follows:

$$\begin{aligned} &X_{b1}^{k+1} \in \{X_{b1}^{k+1} + \varepsilon D_{b1}^{k+1}\}, \text{s.t. } f(X_{b1}^{k+1}) = \max\{f(X_{b1}^{k+1} + \varepsilon D_{b1}^{k+1})\}, \\ &X_{b2}^{k+1} \in \{X_{b2}^{k+1} + \varepsilon D_{b2}^{k+1}\}, \text{s.t. } f(X_{b2}^{k+1}) = \max\{f(X_{b2}^{k+1} + \varepsilon D_{b2}^{k+1})\}, \\ &\varepsilon = \frac{(1001 - j)||L^0||_2}{2000\ln(k+2)}, \end{aligned} \tag{7}$$

where ε is a set of search steps (a group of values, the form adopted in this paper is listed above), $j = 1, 2, \ldots, 2000$, and $||x||_2$ is the Euclidean distance of x.

2.5 Creating of better member

The optimization strategy above is mainly designed according to the behavior of male lions in the pride. This analysis suggested that female density, cub productivity, adult female mortality, and the average age of adult females determine the long-term persistence of a lion pride [32]. Thus, if a lion pride has not evolved for a long period, it may disappear and all territory and resources will be occupied by other stronger prides. Hence in LPO, if the evolution of a pride stops for a period (bigger than a threshold $thls$, called long stagnation, and $thls$ is bigger than ths), a stronger best member is created to represent the stronger pride, and the stronger best member is created by one-dimension search in each dimension (see formula (8)).

$$\begin{aligned} &X_{b1}^{k+1} \in \{X_{b1}^{k+1} + \varepsilon e(n,i)\}, \text{s.t.}, \ f(X_{b1}^{k+1}) = \max\{f(X_{b1}^{k+1} + \varepsilon e(n,i)), i = 1, 2, \ldots, n\}, \\ &\varepsilon = \frac{L^0(1)(201 - j)}{200 \times 10^{2tls-1}}, j = 1, 2, \ldots, 400, \end{aligned} \tag{8}$$

where ε is a set of search steps (a group of value, the form adopted in this paper is listed above), tls is the long stagnation times, and $e(n,i)$ is a n-dimension unit vector with just one nonzero element in ith position.

The flowchart of the LPO algorithm is presented in Figure 2. The pseudocode for the LPO algorithm is listed in Table 1.

3 Simulation results

3.1 Test functions

It is thought that no algorithm can have elevated performance on all problems [33]; thus to have an all-sided understanding of the capability of our proposed LPO, we adopted a large set of benchmark functions as listed in Table 2 [2,7]. The 23 benchmark functions can be grouped into unimodal functions (f_1–f_7), many local minima multimodal functions (f_8–f_{13}), and few local minima multimodal functions (f_{14}–f_{23}). Though they were adopted (or partly adopted) by many other researches [2,5,7,23,28,29], their dimensions are small (up to 30) compared with real-world optimization problems [2]. Many local minima

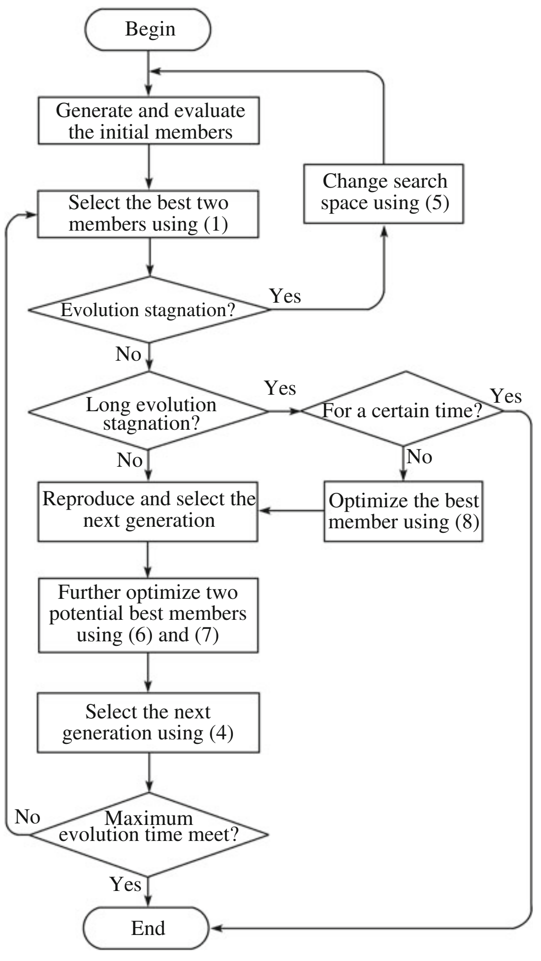
Begin
Generate and evaluate the initial members
Select the best two members using (1)
Change search space using (5)
Evolution stagnation?
Yes
No
Long evolution stagnation?
Yes
For a certain time?
Yes
No
No
Reproduce and select the next generation
Optimize the best member using (8)
Further optimize two potential best members using (6) and (7)
Select the next generation using (4)
No
Maximum evolution time meet?
Yes
End

Figure 2 Flowchart of the LPO algorithm.

multimodal functions (f_8–f_{13}) are regarded as the most difficult functions to optimize, since the number of local minima increases exponentially as the function dimension increases [34]. Thus, to evaluate the performance of LPO on solving large-scale optimization problems, multimodal functions f_8–f_{13} are selected and extended to 300 dimensions as listed in Table 3. For detailed information, please shift to [7].

3.2 Experimental settings

The parameters of LPO are set as follows: The initial population of LPO is generated uniformly at random in the search space, and the population is set to 50. The basic mating coefficient mc_0 is 2. The constants k_1, k_2 used in changing search space are set to 1000 and 5, respectively. Stagnation threshold ths and long stagnation threshold $thls$ are set to 10 and 100 for complicated problems (function evolutions>100000), 5 and 20 for general problems (10000⩽function evolutions⩽100000), and 2 and 4 for simple problems (function evolutions<10000) respectively. If long stagnation happens more than 4 times (named tolerance of long stagnation times), the optimization will terminate. In the comparison of best members in the current generation and the last generation, just when $f(X_{b1})^{k+1} > f(X_{b1}^k) + 0.0001$, the stagnation times ts will be set to 0. The settings of these parameters are used throughout all of our experiments.

We compared the performance of LPO with that of five different EAs:

- genetic algorithm (GA)[4];
- evolutionary programming (EP) [35,36];
- evolution strategies (ES) [37];
- particle swarm optimization (PSO) [38];
- group search optimizer (GSO) [2].

Table 1 Pseudocode for the LPO algorithm

```
Set the generation $k = 1$;
Generate the initial members randomly in the search space and calculate the fitness of each member $f(X_i^k)$;
WHILE (the termination conditions are not met);
        Surviving of two best members    Select the best two members of current generation using formula (1);
                                         Replace worst member of this generation with the saved best member;
                                         If the best member is better than that in $k-1$ generation, update the
                                         saved two best members;
                                         Else replace the saved second best member with the best member in
                                         this generation;
        Scaling of searching space       If stagnation times are over the threshold $ths$, change the search space
                                         using (5);
        Update the pride                 Generate the initial members randomly in the search space and calcu-
                                         late the fitness of each member $f(X_i^k)$;
                                         Continue;
        Creating of better member        If stagnation times are over the bigger threshold $thls$, Optimize the best
                                         member by one-dimension search in each dimension using (8);
                                         If this case appears for several times, terminate the optimization and
                                         exit the program;
    FOR (each member in the pride);
        Novel crossover strategy         Randomly create and calculate mating coefficient $mc_i^k$ using (3);
                                         For (each member in the pride), reproduce four children for each pair
                                         using (2);
                                         Select the potentially best evolution directions using (6);
                                         Search in the potentially best evolution directions for better member
                                         using (7);
                                         Select the best members to the next generation using (4);
                                         Set $k = k + 1$;
    END FOR;
END WHILE;
```

Among published results on the five different EAs, we adopt the test results of $f_1 - f_{23}$ from [2,7,28] for comparison. In [7,28], FEP and FES, which replace Gaussian mutations of conventional EP (CEP) and conventional ES (CES) with Cauchy mutations, were proposed and the published results are adopted for comparison. See [2,7,28] for the parameters of these algorithms. For the 300-dimensional cases, the published results in [2] are adopted for comparison, and the parameters of LPO are the same as those used in 30-dimensional problems.

To be equal to the run times adopted for the reference EAs in previous studies, one thousand independent runs of LPO algorithm have been executed on 30-dimensional benchmark functions f_1–f_{13}, for benchmark functions f_{14}–f_{23}, 50 runs have been executed. The numbers of function evaluations for the 23 benchmark functions are tabulated in Table 4. It can be seen from the table that the numbers of function evaluations of LPO are less than or equal to other EAs. For 300-dimensional benchmark functions, five runs were executed to obtain the average results. The maximum number of function evaluations for the six 300-dimensional benchmark functions is set to 3750000 for LPO, which is the same as the other four algorithms [2].

The experiments include an average test on all algorithms for each benchmark function. For 30-dimensional benchmark functions f_1–f_{13}, which are executed for one thousand independent runs, the standard deviation test is adopted.

The experiments were carried out on a PC with a 1.80-GHz Inter Processor and 1.0-GB RAM. All the programs were written and executed in MATLAB 6.5. The operating system was Microsoft Windows XP. The qualifications were the same as [2]. The parameters of MATLAB 6.5 were set to the default values to ensure the experimental condition was worse or equal to that in [2], in order to get a comparable CPU time.

Table 2 Twenty three benchmark functions. n is the dimension of the function, S is the feasible search space, $f_{\min}$ is the global minimum value of the function, and $X_{\min}$ is the location of global minimum value of the function. Please refer to [7] for the values of parameters in f_{14}–f_{15}, f_{19}–f_{23}

Test function	n	S	$f_{\min}$	$X_{\min}$
$f_1(x)=\sum_{i=1}^{n} x_i^2$	30	$[-100,100]^n$	0	$(0,\dots,0)$
$f_2(x)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	30	$[-10,10]^n$	0	$(0,\dots,0)$
$f_3(x)=\sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2$	30	$[-100,100]^n$	0	$(0,\dots,0)$
$f_4(x)=\max_i\{\lvert x_i\rvert, 1\leqslant i\leqslant n\}$	30	$[-100,100]^n$	0	$(0,\dots,0)$
$f_5(x)=\sum_{i=1}^{n-1}(100(x_{i+1}-x_i^2)^2+(x_i-1))^2$	30	$[-30,30]^n$	0	$(1,\dots,1)$
$f_6(x)=\sum_{i=1}^{n}(\lfloor x_i+0.5\rfloor)^2$	30	$[-100,100]^n$	0	$(0,\dots,0)$
$f_7(x)=\sum_{i=1}^{n} ix_i^4+\text{random}[0,1)$	30	$[-1.28,1.28]^n$	0	$(0,\dots,0)$
$f_8(x)=\sum_{i=1}^{n}(x_i\sin\sqrt{\lvert x_i\rvert})^2$	30	$[-500,500]^n$	-12569.5	$(420.9687,\dots,420.9687)$
$f_9(x)=\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i)+10)^2$	30	$[-5.12,5.12]^n$	0	$(0,\dots,0)$
$f_{10}(x)=-20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2})-\exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))+20+e$	30	$[-32,32]^n$	0	$(0,\dots,0)$
$f_{11}(x)=\frac{1}{4000}\sum_{i=1}^{n}(x_i)^2-\prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}})+1$	30	$[-600,600]^n$	0	$(100,\dots,100)$
$f_{12}(x)=\frac{\pi}{n}\{10\sin^2(\pi y_1)+\sum_{i=1}^{n-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]+(y_n-1)^2\}+\sum_{i=1}^{n}u(x_i,10,100,4)$ $y_i=1+(x_i+1)/4$ $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a\\ 0, & -a\leqslant x_i\leqslant a\\ k(-x_i-a)^m, & x_i<-a\end{cases}$	30	$[-50,50]^n$	0	$(1,\dots,1)$
$f_{13}(x)=0.1\{\sin^2(\pi 3x_i)+\sum_{i=1}^{n-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})]+(x_n-1)^2[1+\sin^2(2\pi x_n)]\}+\sum_{i=1}^{n}u(x_i,5,100,4)$	30	$[-50,50]^n$	0	$(1,\dots,1)$
$f_{14}(x)=[\frac{1}{500}+\sum_{i=1}^{25}\frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})^6}]^{-1}$	2	$[-65.5,65.5]^n$	1	$(-32,-32)$
$f_{15}(x)=\sum_{i=1}^{11}[a_i-\frac{x_1(b_i^2+b_ix_i)}{b_i^2+b_ix_3+x_4}]^2$	4	$[-5,5]^n$	3.08×10^{-4}	$(0.1928,0.1908,0.1231,0.1358)$
$f_{16}(x)=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4$	2	$[-5,5]^n$	-1.0316	$(0.08983,-0.7126)$, $(-0.08983,0.7126)$
$f_{17}(x)=(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}-6)^2+10(1-\frac{1}{8\pi})\cos x_1+10$	2	$[-5,10]\times[0,15]$	0.398	$(-3.142,2.275),(3.142,2.275),(9.425,2.425)$
$f_{18}(x)=[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)]\times[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)]$	2	$[-2,2]^n$	3	$(0,-1)$
$f_{19}(x)=-\sum_{i=1}^{4}c_i\exp[-\sum_{j=1}^{n}a_{ij}(x_j-p_{ij})^2]$	3	$[0,1]^n$	-3.86	$(0.114,0.556,0.852)$
$f_{20}(x)=-\sum_{i=1}^{4}c_i\exp[-\sum_{j=1}^{n}a_{ij}(x_j-p_{ij})^2]$	6	$[0,1]^n$	-3.32	$(0.201,0.150,0.477,0.275,0.311,0.657)$
$f_{21}(x)=-\sum_{i=1}^{5}[(x-a_i)(x-a_i)^{\mathrm{T}}+c_i]^{-1}$	4	$[0,10]^n$	-10	$(4,4,4,4)$
$f_{22}(x)=-\sum_{i=1}^{7}[(x-a_i)(x-a_i)^{\mathrm{T}}+c_i]^{-1}$	4	$[0,10]^n$	-10	$(4,4,4,4)$
$f_{23}(x)=-\sum_{i=1}^{10}[(x-a_i)(x-a_i)^{\mathrm{T}}+c_i]^{-1}$	4	$[0,10]^n$	-10	$(4,4,4,4)$

Table 3 Six 300-dimensional multimodal benchmark functions

Test function	n	S	$f_{\min}$	$X_{\min}$
$f_8(x)^{300}$	300	$[-500,500]^n$	-125694.7	$(420.9687,\dots,420.9687)$
$f_9(x)^{300}$	300	$[-5.12,5.12]^n$	0	$(0,\dots,0)$
$f_{10}(x)^{300}$	300	$[-32,32]^n$	0	$(0,\dots,0)$
$f_{11}(x)^{300}$	300	$[-600,600]^n$	0	$(100,\dots,100)$
$f_{12}(x)^{300}$	300	$[-50,50]^n$	0	$(0,\dots,0)$
$f_{13}(x)^{300}$	300	$[-50,50]^n$	0	$(0,\dots,0)$

Table 4 Maximal number of function evolutions for function f_1–f_{23}

	LPO	GSO/GA/PSO	CEP/FEP/CES/FES		LPO	GSO/GA/PSO	CEP/FEP/CES/FES
f_1	150000	150000	150000	f_{13}	150000	150000	150000
f_2	150000	150000	200000	f_{14}	1000	7500	10000
f_3	150000	250000	500000	f_{15}	250000	250000	400000
f_4	100000	150000	500000	f_{16}	750	1250	10000
f_5	100000	150000	2000000	f_{17}	1250	5000	10000
f_6	150000	150000	150000	f_{18}	5000	10000	10000
f_7	150000	150000	300000	f_{19}	750	4000	10000
f_8	150000	150000	900000	f_{20}	2500	7500	20000
f_9	250000	250000	500000	f_{21}	7500	10000	10000
f_{10}	150000	150000	150000	f_{22}	7500	10000	10000
f_{11}	150000	150000	200000	f_{23}	7500	10000	10000
f_{12}	150000	150000	150000				

Table 5 Comparison of LPO with GSO, GA, and PSO on benchmark function f_1–f_7. All results were averaged over 1000 runs

	Algorithms	LPO	GSO	PSO	GA
f_1	Mean (Rank)	2.5007×10^{-12} (2)	1.9481×10^{-8} (3)	3.6927×10^{-37} (1)	3.1711 (4)
	Std. (CPU)	4.0345×10^{-13} (11.9)	1.1629×10^{-8} (27.6)	2.4598×10^{-36} (36.3)	1.6621 (12.5)
f_2	Mean (Rank)	7.5182×10^{-7} (2)	3.7039×10^{-5} (3)	2.9168×10^{-24} (1)	0.5711 (4)
	Std. (CPU)	7.9940×10^{-8} (15.4)	8.6185×10^{-5} (29.0)	1.1362×10^{-23} (38.5)	0.1326 (13.1)
f_3	Mean (Rank)	6.7303×10^{-4} (1)	5.5829 (3)	1.1979×10^{-3} (2)	9769.9145 (4)
	Std. (CPU)	2.3793×10^{-4} (39.0)	3.6813 (24.5)	2.1109×10^{-3} (36.4)	2594.9593 (12.8)
f_4	Mean (Rank)	0.0266 (1)	0.1078 (2)	0.4123 (3)	7.961 (4)
	Std. (CPU)	0.0050 (20.1)	3.9981×10^{-2} (27.4)	0.25 (36.7)	1.5063 (12.1)
f_5	Mean (Rank)	3.1034×10^{-4} (1)	49.8359 (3)	37.3582 (2)	338.5616 (4)
	Std. (CPU)	1.7183×10^{-4} (35.3)	30.1771 (27.8)	32.1436 (37.6)	361.497 (11.3)
f_6	Mean (Rank)	0 (1)	1.600×10^{-2} (2)	0.146 (3)	3.697 (4)
	Std. (CPU)	0 (5.6)	0.1333 (27.1)	0.4182 (38.9)	1.9517 (12.3)
f_7	Mean (Rank)	0.5000 (4)	7.3773×10^{-2} (2)	9.9024×10^{-3} (1)	0.1045 (3)
	Std. (CPU)	0.2897 (28.4)	9.2557×10^{-2} (31.4)	3.5380×10^{-2} (41.4)	0.0362 (12.2)
Averaged rank		1.7	2.6	1.9	3.9
Final rank		1	3	2	4

3.3 Unimodal functions

It is proved that many deterministic optimization algorithms that use gradient information have good performance in solving unimodal problems. However, the test of EAs on unimodal functions is needed to have a comprehensive evaluation of an algorithm's capability, and unimodal functions have been adopted to assess the convergence rates of EAs [28]. Thus we tested the LPO on a set of unimodal functions in comparison with other EAs. Table 5 lists the means and standard deviations of the function values found in 1000 runs, and the average CPU time in seconds for each algorithm on unimodal functions f_1–f_7. The results generated from CEP, FEP, CES, FES and GSO are tabulated in Table 6 in comparison with the results generated by our LPO algorithm (as GSO was compared in both aspects in [2], we also adopted GSO both in the two comparisons in order to obtain a comparable rank in the following parts).

Table 5 reveals that the LPO generates significantly better results than GA and GSO on the unimodal functions f_1–f_7 on the whole. For the comparison of LPO and PSO, LPO outperforms PSO slightly on the whole, PSO has better performance in f_1, f_2, and f_7, while LPO performs better in f_3–f_6. In summary, the four algorithms can be ranked as LPO>PSO>GSO>GA according to their test performances. From

Table 6 Comparison of LPO with GSO, CEP, FEP, CES, and FES on benchmark function $f_1 - f_7$

	Mean function value (Rank) (Standard deviation)					
Function	LPO	GSO	FEP	CEP	FES	CES
f_1	2.5×10^{-12} (1)	1.9×10^{-8} (2)	5.7×10^{-4} (6)	2.2×10^{-4} (4)	2.5×10^{-4} (5)	3.4×10^{-5} (3)
	(4.0×10^{-13})	(1.2×10^{-8})	(1.3×10^{-4})	(5.9×10^{-4})	(6.8×10^{-4})	(8.6×10^{-6})
f_2	7.5×10^{-7} (1)	3.7×10^{-5} (2)	8.1×10^{-3} (4)	2.6×10^{-3} (3)	6.0×10^{-2} (6)	2.1×10^{-2} (5)
	(8.0×10^{-8})	(8.6×10^{-5})	(7.7×10^{-4})	(1.7×10^{-4})	(9.6×10^{-3})	(2.2×10^{-3})
f_3	6.7×10^{-4} (2)	5.8 (6)	1.6×10^{-2} (4)	5.0×10^{-2} (5)	1.4×10^{-3} (3)	1.3×10^{-4} (1)
	(2.4×10^{-4})	(3.7)	(1.4×10^{-2})	(6.6×10^{-2})	(5.3×10^{-4})	(8.5×10^{-5})
f_4	0.0266 (2)	0.1078 (3)	0.3 (4)	2.0 (6)	5.5×10^{-3} (1)	0.35 (5)
	(0.0050)	(0.0400)	(0.5)	(1.2)	(6.5×10^{-4})	(0.42)
f_5	3.1×10^{-4} (1)	49.84 (6)	5.06 (2)	6.17 (3)	33.28 (5)	6.69 (4)
	(1.7×10^{-4})	(30.18)	(5.87)	(13.61)	(43.13)	(14.45)
f_6	0 (1)	1.60×10^{-2} (4)	0 (1)	577.76 (6)	0 (1)	411.16 (5)
	(0)	(0.13)	(0)	(1125.76)	(0)	(695.35)
f_7	0.5 (6)	7.4×10^{-2} (5)	7.6×10^{-3} (1)	1.8×10^{-2} (3)	1.2×10^{-2} (2)	3.0×10^{-2} (4)
	(0.3)	(9.3×10^{-2})	(2.6×10^{-3})	(6.4×10^{-3})	(5.8×10^{-3})	(1.5×10^{-2})
Averaged rank	2.0	4.0	3.1	4.3	3.3	3.9
Final rank	1	5	2	6	3	4

Table 7 Comparison of LPO with GSO, GA, and PSO on benchmark function f_8–f_{13}. All results were averaged over 1000 runs

	Algorithms	LPO	GSO	PSO	GA
f_8	Mean(Rank)	−12569.4866 (2)	−12569.4882 (1)	−9659.6993 (4)	−12566.0977 (3)
	Std. (CPU)	1.3059×10^{-10} (40.6)	2.2140×10^{-2} (32.7)	463.725 (43.0)	2.1088 (16.2)
f_9	Mean(Rank)	9.8517×10^{-26} (1)	1.0179 (3)	20.7863 (4)	0.6509 (2)
	Std. (CPU)	2.3185×10^{-26} (13.9)	0.9509 (50.1)	5.9400 (71.4)	0.3594 (17.4)
f_{10}	Mean(Rank)	3.6714×10^{-7} (1)	2.6548×10^{-5} (2)	1.3404×10^{-3} (3)	0.8678 (4)
	Std. (CPU)	3.0267×10^{-8} (18.1)	3.0820×10^{-5} (34.4)	4.2388×10^{-2} (44.1)	0.2805(19.3)
f_{11}	Mean(Rank)	5.9388×10^{-12} (1)	3.0792×10^{-2} (2)	0.2323 (3)	1.0038 (4)
	Std. (CPU)	1.6813×10^{-12} (26.1)	3.0867×10^{-2} (35.9)	0.4434 (45.4)	6.7545×10^{-2} (16.1)
f_{12}	Mean(Rank)	1.7757×10^{-14} (1)	2.7648×10^{-11} (2)	3.9503×10^{-2} (3)	4.3572×10^{-2} (4)
	Std. (CPU)	1.2321×10^{-14} (51.0)	9.1674×10^{-11} (34.4)	9.1424×10^{-2} (44.8)	5.0579×10^{-2} (23.2)
f_{13}	Mean(Rank)	2.4583×10^{-13} (1)	4.6948×10^{-5} (2)	5.0519×10^{-2} (3)	0.1681 (4)
	Std. (CPU)	1.7165×10^{-13} (50.4)	7.0100×10^{-4} (35.6)	0.5691 (45.5)	7.0681×10^{-2} (22.6)
Averaged rank		1.17	2	3.33	3.5
Final rank		1	2	3	4

this table, we can also find that the average CPU time required for LPO (22.2 s) is less than those of PSO (38.0 s) and GSO (27.8 s), but is more than GA (12.3 s).

It is obvious from Table 6 that LPO ranks the first among the six algorithms, because it significantly outperforms all of the other five algorithms, and LPO requires much fewer function evolutions than CEP, FEP, CES, and FES.

3.4 Multimodal functions with many local minima

3.4.1 30-*dimension functions*

Multimodal Functions are regarded as the most challenging problems, and many researchers developed numerous algorithms to solve these problems. The experimental results of LPO from one thousand runs and the published results of GSO, PSO, and GA are listed in Table 7. The results generated from CEP,

Table 8 Comparison of LPO with GSO, CEP, FEP, CES, and FES on benchmark function f_8–f_{13}

	Mean function value (Rank) (Standard deviation)					
Function	LPO	GSO	FEP	CEP	FES	CES
f_8	−12569 (2)	−12569 (1)	−12555 (4)	−7917 (5)	−12556 (3)	−7550 (6)
	(1.3×10^{-10})	(2.2×10^{-2})	(52.6)	(634.5)	(32.53)	(631.39)
f_9	9.9×10^{-26} (1)	1.02 (4)	4.6×10^{-2} (2)	89.0 (6)	0.16 (3)	70.82 (5)
	(2.3×10^{-26})	(0.95)	(1.2×10^{-2})	(23.1)	(0.33)	(21.49)
f_{10}	3.7×10^{-7} (1)	2.7×10^{-5} (2)	1.8×10^{-2} (4)	9.2 (6)	1.2×10^{-2} (3)	9.07 (5)
	(3.0×10^{-8})	(3.1×10^{-5})	(2.1×10^{-2})	(2.8)	(1.8×10^{-3})	(2.84)
f_{11}	5.9×10^{-12} (1)	3.1×10^{-2} (3)	1.6×10^{-2} (2)	8.6×10^{-2} (5)	3.7×10^{-2} (4)	0.38 (6)
	(1.7×10^{-12})	(2.9×10^{-2})	(2.2×10^{-2})	(0.12)	(5.0×10^{-2})	(0.77)
f_{12}	1.8×10^{-14} (1)	2.8×10^{-11} (2)	9.2×10^{-6} (3)	1.76 (6)	2.8×10^{-2} (4)	1.18 (5)
	(1.2×10^{-14})	(9.2×10^{-11})	(6.1×10^{-5})	(2.4)	(8.1×10^{-11})	(1.87)
f_{13}	2.5×10^{-13} (1)	4.7×10^{-5} (2)	1.6×10^{-4} (4)	1.4 (5)	4.7×10^{-5} (3)	1.9 (6)
	(1.7×10^{-13})	(7.0×10^{-4})	(7.3×10^{-5})	(3.7)	(1.5×10^{-5})	(3.33)
Averaged rank	1.17	2.33	3.17	5.5	3.33	5.5
Final rank	1	2	3	5	4	5

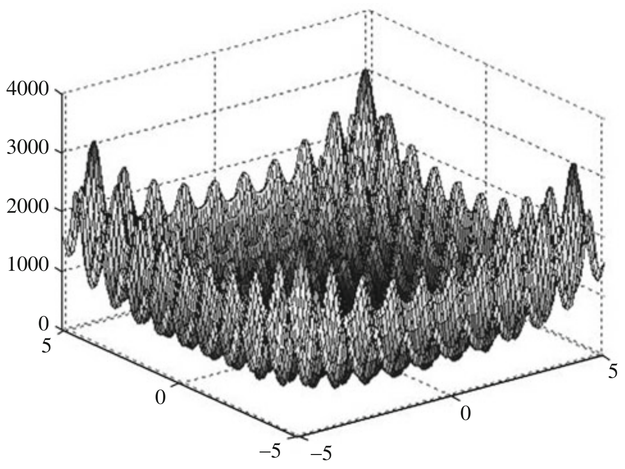

Figure 3 The two-dimensional version of f_9.

FEP, CES, FES and GSO are tabulated in Table 8 for comparison with the results generated from our LPO algorithm.

According to the results in Table 7, it is clear that LPO markedly outperforms GA and PSO in all benchmark functions, LPO also has significantly better performance than GSO on most benchmark functions, and the only exception is the function f_8, thus LPO markedly outperforms GSO, PSO, and GA on the whole. According to the test performances of the four algorithms, they can be ordered as LPO>GSO>PSO>GA. The standard deviations of LPO also reveal that LPO has consistent performance. From this table, we can also find that the average CPU time required for LPO (33.4 s) is less than those of PSO (49.0 s) and GSO (37.2 s), but is more than GA (19.1 s).

Table 8 shows that LPO has significantly better performance than FEP, CEP, FES, CES and GSO on the whole. LPO generates significantly better results than FEP, CEP, FES, and CES on all of the multimodal functions f_8–f_{13}. Taking Generalized Rastrigin's Function f_9 for example, the two-dimensional version of f_9 is shown in Figure 3. Even in two-dimensional version, there are already many local minima, and because the number of local minima increases exponentially as the function dimension increases [34], the optimization results of GSO, PSO, GA, FEP, CEP, FES, and CES on f_9 (30 dimension) are quite large, while LPO obtains an excellent result.

Table 9 Comparison of LPO with GSO, CEP, FEP, CES, and FES on benchmark function $f_8(x)^{300}$–$f_{13}(x)^{300}$

Function	Mean function value (Rank) (Standard deviation)					
	LPO	GSO	GA	PSO	EP	ES
$f_8(x)^{300}$	−125694.9	−125351.2	−117275.3	−87449.2	−78311.9	−66531.3
$f_9(x)^{300}$	1.0113×10^{-24}	98.9	121.3	427.1	383.3	583.2
$f_{10}(x)^{300}$	3.7058×10^{-7}	1.3527×10^{-3}	6.24	3.9540×10^{-6}	0.2946	9.6243
$f_{11}(x)^{300}$	9.3888×10^{-12}	1.8329×10^{-7}	0.37	1.81	2.8244×10^{-2}	0.1583
$f_{12}(x)^{300}$	1.2838×10^{-14}	8.2582×10^{-8}	52.82	14.56	39.3	3093.2
$f_{13}(x)^{300}$	8.8317×10^{-13}	2.0157×10^{-7}	178.34	549.2	738.2	2123.2

3.4.2 300-*dimension functions*

With the development of science and technology, the real world optimization problems we are facing usually include hundreds or even thousands of variables. Thus, the capability of solving high dimension problems is crucial for an algorithm's applicability to real world problems. However, previous studies have shown that although some algorithms generated good results on relatively low-dimensional ($n \leqslant 30$) benchmark problems, they do not perform satisfactorily for some large-scale problems [39]. Therefore, to survey the capability of solving high dimension problems, a set of multimodal benchmark functions (f_8–f_{13}) are extended to 300 dimensions and used in our experimental studies as high dimensional benchmark functions. The experiment results and the published results are gathered and presented in Table 9.

Table 9 reveals that LPO significantly outperforms other five algorithms. For example, LPO optimizes an extremely small result even in 300-dimension f_9, while the other five algorithms perform much worse and are 9.8×10^{25} bigger than LPO at least. It can also be seen that although PSO achieves satisfactory results in 30-dimensional multimodal benchmark problems (see Table 7), it cannot be scaled up to handle most of the 300-dimensional cases except $f_{10}(x)^{300}$. GSO performs consistently from 30-dimensional cases to 300-dimensional cases; however, different from the 30-dimensional cases, LPO outperforms GSO on all 300-dimensional benchmark functions.

A limited scale of research scalability of EAs has been found [39,40]. In [39], four EP algorithms, namely CEP, FEP, improved FEP (IFEP) [7], and a mixed EP (MEP) [39], were studied. The benchmark functions used in their studies were a unimodal function f_1 and a multimodal function f_{10} with dimensions ranging from 100 to 300. It was found that CEP and FEP failed to converge on function f_{10}. The average results obtained by IFEP and MEP on function f_{10} in 300 dimensions were 7.6×10^{-2} and 5.5×10^{-2}, respectively, which were both worse than the results of our LPO algorithm.

Liu and Yao [40] also improved FEP with cooperative coevolution by decoupling the whole optimization function to a set of coordinates of populations. Eight functions, including four unimodal and four multimodal functions were used as benchmark functions in their studies. The results presented in their paper were excellent, e.g., the result on 300-dimensional f_{10} was 3.6×10^{-4}. However, they are still much worse than the results of LPO algorithm.

3.5 Multimodal functions with a few local minima

Though the benchmark functions f_{14}–f_{23} are multimodal, they are in low dimensions ($n \leqslant 6$) and have only a few local minima. Compared to the multimodal functions with many local minima (f_8–f_{13}), this set of functions is not challenging [2]: Some of them can even be solved efficiently by deterministic algorithms [41,42].

According to the comparison results in Table 10, it is clear that LPO ranks first compared with GA, PSO and GSO. In the comparison with GA, LPO achieved better results on all benchmark functions f_{14}–f_{23}. In comparison with PSO, LPO has better performance in f_{14}, f_{16}–f_{21}. In comparison with GSO, LPO achieved the same average results on benchmark functions f_{14}, f_{16}–f_{19}, better results on function f_{20}–f_{23}, and worse result on function f_{15}. According to the test performances of the four algorithms,

Table 10 Comparison of LPO with GSO, GA, and PSO on benchmark function $f_{14} - f_{23}$. All results were averaged over 1000 runs

	Algorithms	LPO	GSO	PSO	GA
f_{14}	Mean(Rank)	0.9980 (1)	0.9980 (1)	1.0239 (4)	0.9989 (3)
	Std. (CPU)	9.7905×10^{-8} (1.6)	0 (1.5)	0.1450 (1.8)	4.4333×10^{-3} (0.9)
f_{15}	Mean(Rank)	5.6188×10^{-4} (3)	3.7713×10^{-4} (1)	3.8074×10^{-4} (2)	7.0878×10^{-3} (4)
	Std. (CPU)	3.2395×10^{-4} (13.2)	2.5973×10^{-4} (36.6)	2.5094×10^{-4} (63.5)	$7.8549\text{E}\times10^{-3}$ (15.3)
f_{16}	Mean(Rank)	−1.031628 (1)	−1.031628 (1)	−1.0160 (4)	−1.0298 (3)
	Std. (CPU)	7.8267×10^{-8} (0.4)	0 (0.2)	1.2786×10^{-2} (0.3)	3.1314×10^{-3} (0.5)
f_{17}	Mean(Rank)	0.3979 (1)	0.3979 (1)	0.4040 (3)	0.4040 (3)
	Std. (CPU)	1.6336×10^{-9} (0.6)	0 (0.7)	6.8805×10^{-2} (1.1)	1.0385×10^{-2} (0.5)
f_{18}	Mean(Rank)	3.0000 (1)	3.0 (1)	3.0050 (3)	7.5027 (4)
	Std. (CPU)	8.2761×10^{-10} (0.6)	0 (1.5)	1.2117×10^{-3} (2.3)	10.3978 (1.0)
f_{19}	Mean(Rank)	−3.8628 (1)	−3.8628 (1)	−3.8582 (4)	−3.8624 (3)
	Std. (CPU)	1.9127×10^{-4} (0.7)	3.8430×10^{-6} (0.5)	3.2127×10^{-3} (0.9)	6.2841×10^{-4} (0.3)
f_{20}	Mean(Rank)	−3.3125 (1)	−3.2697 (2)	−3.1846 (4)	−3.2627 (3)
	Std. (CPU)	3.2417×10^{-2} (1.7)	5.9647×10^{-2} (1.0)	6.1053×10^{-2} (1.8)	6.0398×10^{-2} (0.7)
f_{21}	Mean(Rank)	−7.7062 (1)	−6.09 (3)	−7.5439 (2)	−5.1653 (4)
	Std. (CPU)	2.5598 (1.9)	3.4563 (1.7)	3.0303 (2.5)	2.9254 (1.3)
f_{22}	Mean(Rank)	−7.4658 (2)	−6.5546 (3)	−8.3553 (1)	−5.4432 (4)
	Std. (CPU)	2.8716 (1.9)	3.2443 (1.8)	2.0184 (2.6)	3.2778 (1.3)
f_{23}	Mean(Rank)	−8.3786 (2)	−7.4022 (3)	−8.9439 (1)	−4.9108 (4)
	Std. (CPU)	2.7921 (2.0)	3.2131 (1.9)	1.6304 (2.8)	3.4871 (1.4)
Averaged rank		1.4	1.7	2.8	3.5
Final rank		1	2	3	4

they can be ordered as LPO>GSO>PSO>GA. From this table, we can also find that the average CPU time required for LPO (2.5 s) is less than those of PSO (8.0 s) and GSO (4.7 s), but is more than GA (2.3 s).

Table 11 reveals that LPO generates significantly better results than FES and FEP on most benchmark functions except f_{15} and f_{18}, and better results than CEP and CES on most benchmark functions except f_{15}, f_{18}, f_{22} and f_{23}. And the six algorithms are ranked as LPO>GSO>CEP>CES>FES>FEP.

4 Algorithm property

4.1 Investigation of the effects of control parameters

For all experiments above, a certain set of parameter values were used. Here in this part, we will study how LPO performs with different set of parameter values. Since the multimodal functions are regarded as the most challenging problems [2], we are more interested in how LPO performs on multimodal functions and select four multimodal functions, namely, f_8, f_9, f_{12}, and f_{13}. The control parameters of LPO mainly involve constants k_1, k_2 used in changing search space, stagnation threshold ths, basic mating coefficient mc_0, long stagnation threshold $thls$, and tolerance of long stagnation times. We test the LPO algorithm with different parameter values on the four benchmark functions for 50 runs. The results are tabulated in Tables 12–15.

Table 12 indicates that the effect of the constants k_1, k_2 used in changing search space on the performance of LPO is not significant on the whole. For f_8, f_9, f_{12} and f_{13}, the effect of constants k_1, k_2 on the precision and efficiency of the algorithm is really small and can be ignored.

Table 13 reveals that the effects of stagnation threshold ths, and long stagnation threshold $thls$ on the precision of LPO are not significant, and that the main effects of the two parameters on the LPO algorithm are the efficiency. When ths is relatively big, e.g., 100 or even bigger, the efficiency of LPO is

Table 11 Comparison of LPO with GSO, CEP, FEP, CES, and FES on benchmark function $f_{14} - f_{23}$

	Mean function value (Rank) (Standard deviation)					
Function	LPO	GSO	FEP	CEP	FES	CES
f_{14}	0.9980 (1) (9.8×10^{-8})	0.9980 (1) (0)	1.22 (4) (0.56)	1.66 (5) (1.19)	1.20 (3) (0.63)	2.16 (6) (1.82)
f_{15}	5.6188×10^{-4} (4) (3.2×10^{-4})	4.1687×10^{-4} (1) (3.1×10^{-4})	5.0×10^{-4} (3) (3.2×10^{-4})	4.7×10^{-4} (2) (3.0×10^{-4})	9.7×10^{-4} (5) (4.2×10^{-4})	1.2×10^{-3} (6) (1.6×10^{-5})
f_{16}	-1.031628 (1) (7.8×10^{-8})	-1.031628 (1) (0)	-1.03 (5) (4.9×10^{-4})	-1.03 (5) (4.9×10^{-4})	-1.0316 (3) (6.0×10^{-7})	-1.0316 (3) (6.0×10^{-7})
f_{17}	0.3979 (1) (1.6×10^{-9})	0.3979 (1) (0)	0.398 (3) (1.5×10^{-7})	0.398 (3) (1.5×10^{-7})	0.398 (3) (6.0×10^{-8})	0.398 (3) (6.0×10^{-8})
f_{18}	3.0 (1) (8.3×10^{-10})	3.0 (1) (0)	3.02 (6) (0.11)	3.0 (1) (0)	3.0 (1) (0)	3.0 (1) (0)
f_{19}	-3.8628 (1) (1.9×10^{-4})	-3.8628 (1) (3.8×10^{-6})	-3.86 (3) (1.4×10^{-5})	-3.86 (3) (1.4×10^{-2})	-3.86 (3) (4.0×10^{-3})	-3.86 (3) (1.4×10^{-5})
f_{20}	-3.3125 (1) (3.2×10^{-2})	-3.2697 (4) (6.0×10^{-2})	-3.27 (3) (5.9×10^{-2})	-3.28 (2) (5.8×10^{-2})	-3.23 (6) (0.12)	-3.24 (5) (5.7×10^{-2})
f_{21}	-7.7062 (1) (2.5598)	-6.09 (4) (3.4563)	-5.52 (6) (1.59)	-6.86 (3) (2.67)	-5.54 (5) (1.82)	-6.96 (2) (3.10)
f_{22}	-7.4658 (3) (2.8716)	-6.5546 (5) (3.2443)	-5.52 (6) (2.12)	-8.27 (2) (2.95)	-6.67 (4) (3.01)	-8.31 (1) (3.10)
f_{23}	-8.3786 (3) (2.7921)	-7.4022 (5) (3.2131)	-6.57 (6) (3.14)	-9.10 (1) (2.92)	-7.63 (4) (3.27)	-8.50 (2) (1.25)
Averaged rank	1.7	2.4	4.5	2.7	3.7	3.2
Final rank	1	2	6	3	5	4

Table 12 Mean best function values and standard deviations of function f_8, f_9, f_{12} and f_{13} with different constants k_1, k_2 used in changing search space. All results have been averaged over 50 runs

k_1 and k_2		$f_8(x)$ Mean (Std., CPU)	$f_9(x)$ Mean (Std., CPU)	$f_{12}(x)$ Mean (Std., CPU)	$f_{13}(x)$ Mean (Std., CPU)
k_1	10	-12569 (4.4×10^{-12}, 33)	1.0×10^{-25} (2.0×10^{-26}, 17)	1.6×10^{-14} (1.1×10^{-14}, 48)	3.0×10^{-13} (2.0×10^{-13}, 50)
	1×10^2	-12569 (4.5×10^{-12}, 34)	1.0×10^{-25} (2.5×10^{-26}, 15)	1.8×10^{-14} (1.3×10^{-14}, 50)	2.8×10^{-13} (1.5×10^{-13}, 52)
	1×10^3	-12569 (4.7×10^{-12}, 36)	1.0×10^{-25} (2.1×10^{-26}, 14)	1.8×10^{-14} (1.2×10^{-14}, 50)	2.5×10^{-13} (1.7×10^{-13}, 50)
	1×10^4	-12569 (4.8×10^{-12}, 33)	1.1×10^{-25} (2.6×10^{-26}, 14)	1.8×10^{-13} (1.1×10^{-14}, 50)	2.8×10^{-13} (1.9×10^{-13}, 54)
	1×10^5	-12569 (4.8×10^{-12}, 31)	1.0×10^{-25} (2.7×10^{-26}, 16)	1.8×10^{-14} (1.2×10^{-14}, 54)	1.9×10^{-13} (1.2×10^{-13}, 59)
k_2	3	-12569 (4.0×10^{-12}, 34)	1.0×10^{-25} (2.5×10^{-26}, 14)	1.7×10^{-14} (1.2×10^{-14}, 49)	2.1×10^{-13} (1.3×10^{-13}, 49)
	5	-12569 (4.7×10^{-12}, 36)	1.0×10^{-25} (2.6×10^{-26}, 14)	1.8×10^{-14} (1.2×10^{-14}, 50)	2.5×10^{-13} (1.7×10^{-13}, 50)
	10	-12569 (5.3×10^{-12}, 34)	1.0×10^{-25} (2.3×10^{-26}, 14)	2.0×10^{-14} (1.3×10^{-14}, 50)	2.6×10^{-13} (1.5×10^{-13}, 49)
	50	-12569 (4.7×10^{-12}, 34)	1.0×10^{-25} (2.9×10^{-26}, 14)	2.0×10^{-14} (1.3×10^{-14}, 50)	2.7×10^{-13} (1.5×10^{-13}, 52)

Table 13 Mean best function values and standard deviations of function f_8, f_9, f_{12} and f_{13} with different stagnation thresholds *ths* and long stagnation thresholds *thls* used in changing search space. All results have been averaged over 50 runs

ths		$f_8(x)$	$f_9(x)$	$f_{12}(x)$	$f_{13}(x)$
and	*thls*	Mean (Std., CPU)	Mean (Std., CPU)	Mean (Std., CPU)	Mean (Std., CPU)
ths	5	-12569 $(4.5\times10^{-12}, 38)$	1.1×10^{-25} $(3.1\times10^{-26}, 15)$	1.9×10^{-14} $(1.2\times10^{-14}, 50)$	2.7×10^{-13} $(1.6\times10^{-13}, 52)$
	10	-12569 $(4.7\times10^{-12}, 36)$	1.0×10^{-25} $(3.0\times10^{-26}, 13)$	1.8×10^{-14} $(1.3\times10^{-14}, 49)$	2.5×10^{-13} $(1.7\times10^{-13}, 50)$
	20	-12569 $(4.1\times10^{-12}, 34)$	1.0×10^{-25} $(2.0\times10^{-26}, 13)$	1.8×10^{-14} $(1.3\times10^{-14}, 55)$	2.5×10^{-13} $(1.8\times10^{-13}, 51)$
	50	-12569 $(4.4\times10^{-12}, 37)$	1.0×10^{-25} $(2.8\times10^{-26}, 16)$	1.6×10^{-14} $(1.3\times10^{-14}, 58)$	2.2×10^{-13} $(1.6\times10^{-13}, 59)$
	100	-12569 $(5.2\times10^{-12}, 48)$	1.0×10^{-25} $(2.7\times10^{-26}, 22)$	1.6×10^{-14} $(1.2\times10^{-14}, 60)$	2.2×10^{-13} $(1.4\times10^{-13}, 57)$
thls	50	-12569 $(4.5\times10^{-12}, 26)$	1.0×10^{-25} $(2.4\times10^{-26}, 10)$	2.1×10^{-14} $(1.3\times10^{-14}, 38)$	1.9×10^{-13} $(1.5\times10^{-13}, 38)$
	100	-12569 $(4.7\times10^{-12}, 36)$	1.0×10^{-25} $(2.5\times10^{-26}, 14)$	1.7×10^{-14} $(1.3\times10^{-14}, 48)$	2.5×10^{-13} $(1.7\times10^{-13}, 50)$
	200	-12569 $(4.6\times10^{-12}, 51)$	1.0×10^{-25} $(2.4\times10^{-26}, 22)$	1.7×10^{-14} $(1.2\times10^{-14}, 70)$	3.0×10^{-13} $(1.7\times10^{-13}, 74)$
	500	-12569 $(5.2\times10^{-12}, 101)$	1.0×10^{-25} $(2.6\times10^{-26}, 46)$	1.9×10^{-14} $(1.4\times10^{-14}, 133)$	4.4×10^{-11} $(1.9\times10^{-10}, 135)$

Table 14 Mean best function values and standard deviations of function f_8, f_9, f_{12} and f_{13} with different basic mating coefficients mc_0. All results have been averaged over 50 runs

mc_0	$f_8(x)$	$f_9(x)$	$f_{12}(x)$	$f_{13}(x)$
	Mean (Std., CPU)	Mean (Std., CPU)	Mean (Std., CPU)	Mean (Std., CPU)
0.1	-12569 $(7.9\times10^{-4}, 89)$	1.0×10^{-25} $(2.6\times10^{-26}, 15)$	2.0×10^{-14} $(1.3\times10^{-14},54)$	2.7×10^{-13} $(1.7\times10^{-13}, 68)$
1	-12569 $(4.1\times10^{-12}, 39)$	1.1×10^{-25} $(2.5\times10^{-26}, 14)$	2.0×10^{-14} $(1.3\times10^{-14}, 58)$	2.4×10^{-13} $(1.9\times10^{-13}, 67)$
2	-12569 $(4.7\times10^{-12}, 36)$	1.0×10^{-25} $(2.3\times10^{-26}, 14)$	1.6×10^{-14} $(1.1\times10^{-14}, 48)$	2.5×10^{-13} $(1.7\times10^{-13}, 50)$
5	-12569 $(4.8\times10^{-12}, 42)$	1.0×10^{-25} $(2.5\times10^{-26}, 9)$	1.7×10^{-14} $(1.2\times10^{-14}, 34)$	3.0×10^{-13} $(1.9\times10^{-13}, 21)$
10	-12569 $(4.2\times10^{-2}, 97)$	1.0×10^{-25} $(2.6\times10^{-26}, 9)$	2.0×10^{-14} $(1.3\times10^{-14}, 56)$	2.4×10^{-13} $(1.6\times10^{-13}, 22)$

somewhat low. While *thls* is smaller, the efficiency of LPO is higher. In the experiments, two termination conditions are set, one is to meet the maximum function evolutions, and the other is to meet the tolerance of long stagnation times. Thus, *thls* has significant effect on the efficiency of LPO. Therefore, to obtain a faster algorithm, comparatively low *ths* and *thls* can be set. And in the experiments of this study, *ths* and *thls* are set to 10 and 100 for complicated problems respectively.

Table 14 reveals that the effects of basic mating coefficient mc_0 on the precision of LPO are not significant. For f_9 and f_{13}, a bigger mc_0 leads to a better efficiency and nearly the same precision. For f_8 and f_{12}, too lager or too small mc_0 both lead to a fall in the efficiency, indicating that a proper mc_0 is somewhat better. Thus, the bound of 2–5 is recommended, and in the experiments of this study, mc_0 is set to 2.

Table 15 indicates that the effect of tolerance of long stagnation times is significant both on the precision and the efficiency of LPO algorithm. For small tolerance of long stagnation times (e.g., 1), the evolution may stop too early, leading to an unsatisfactory result. Of course, an early stop of the evolution leads to a higher efficiency of the algorithm. However, too big tolerance of long stagnation times (e.g., 8) leads to high precision but low efficiency; therefore, a moderate value of tolerance of long stagnation times is recommended, and in the experiments of this study, tolerance of long stagnation times is set to 4.

Table 15 Mean best function values and standard deviations of function f_8, f_9, f_{12} and f_{13} with different tolerance of long stagnation times. All results have been averaged over 50 runs

Tolerance	$f_8(x)$	$f_9(x)$	$f_{12}(x)$	$f_{13}(x)$
of *thls*	Mean (Std., CPU)	Mean (Std., CPU)	Mean (Std., CPU)	Mean (Std., CPU)
1	-12563 (1.1, 15)	2.7×10^{-10} (1.5×10^{-10}, 7)	1.8×10^{-4} (2.9×10^{-4}, 34)	7.3×10^{-3} (6.6×10^{-3}, 35)
2	-12569 (1.3×10^{-4}, 27)	1.0×10^{-10} (4.9×10^{-11}, 9)	1.1×10^{-6} (9.0×10^{-7}, 40)	1.4×10^{-5} (1.1×10^{-5}, 39)
3	-12569 (1.0×10^{-8}, 30)	1.0×10^{-17} (2.3×10^{-18}, 12)	1.7×10^{-10} (1.2×10^{-10}, 47)	1.9×10^{-9} (1.2×10^{-9}, 46)
4	-12569 (4.7×10^{-12}, 36)	9.8×10^{-26} (2.4×10^{-26}, 14)	1.8×10^{-14} (1.1×10^{-14}, 51)	2.5×10^{-13} (1.7×10^{-13}, 50)
5	-12569 (4.8×10^{-12}, 39)	0 (0, 16)	2.0×10^{-18} (1.4×10^{-18}, 57)	2.8×10^{-17} (1.7×10^{-17}, 56)
6	-12569 (2.1×10^{-12}, 44)	0 (0, 18)	1.7×10^{-22} (1.2×10^{-22}, 63)	2.2×10^{-21} (1.4×10^{-21}, 60)
7	-12569 (2.5×10^{-12}, 47)	0 (0, 20)	1.9×10^{-26} (1.2×10^{-26}, 68)	2.7×10^{-25} (1.7×10^{-25}, 66)
8	-12569 (2.0×10^{-12}, 52)	0 (0, 23)	1.6×10^{-30} (1.1×10^{-30}, 73)	2.6×10^{-29} (1.8×10^{-29}, 74)

Table 16 Mean best function values and standard deviations of function f_1, f_9 and f_{12} with different search domains. All results have been averaged over 50 runs

$f_1(x)$(minimum in $[0,0,\ldots,0]$)		$f_9(x)$(minimum in $[0,0,\ldots,0]$)		$f_{12}(x)$(minimum in $[1,1,\ldots,1]$)	
Domain	Mean (Std., CPU)	Domain	Mean (Std., CPU)	Domain	Mean (Std., CPU)
$[-100,100]^n$	2.5×10^{-12} (4.0×10^{-13}, 12)	$[-5.12,5.12]^n$	1.0×10^{-25} (2.4×10^{-26}, 14)	$[-50,50]^n$	1.8×10^{-14} (1.1×10^{-14}, 51)
$[-150,50]^n$	2.5×10^{-12} (3.9×10^{-13}, 14)	$[-7.62,2.62]^n$	1.0×10^{-25} (2.6×10^{-26}, 28)	$[-75,25]^n$	1.7×10^{-14} (1.2×10^{-14}, 67)
$[-198,2]^n$	2.6×10^{-12} (3.5×10^{-13}, 14)	$[-10.12,0.12]^n$	9.0×10^{-26} (2.8×10^{-26}, 32)	$[-98,2]^n$	1.8×10^{-14} (1.2×10^{-14}, 67)

4.2 Central tendency of the algorithm

For all unimodal functions and some multimodal functions, the minimum located in the center of the search domain (Table 2), the initial population is set almost symmetrically around the minimum. Therefore, this may have bias for algorithms which have a central tendency.

In this paper, the central tendency of the algorithm is surveyed by changing the search domain. One unimodal function f_1 and two multimodal functions f_9 and f_{12}, of which the minimum is in the center of the search domain, are selected in the test (Table 16), for example, when the search domain is [−10.12,0.12] for f_9, and its minimum locates in $[0,0,\ldots,0]$, namely, the minimum locates in the edge of the domain.

The results show that there is no significant effect of the search domain movement. For the case that the minimum locates in the edge of the domain, LPO can find even the same result as the case that minimum locates in the center. Thus, we can conclude that there is no central tendency of the algorithm LPO.

4.3 Algorithm aspect responsible for better performance

The results above show better performance of LPO than other algorithms. And the investigation of the effects of control parameters shows that the algorithm performance of LPO is sensitive to stagnation threshold *ths*, long stagnation threshold *thls*, and the tolerance of long stagnation times, which indicates that the pride update strategy contributes to the better performance of LPO.

On the other hand, the brutal competition of individuals may be another factor. In the algorithm, it is realized by optimizing the best member by one-dimension search in each dimension. Hence, the test results of LPO with and without one-dimension search in each dimension are listed in Table 17. The results prove the above deduction. The performance of LPO without one-dimension search is unstable. For f_9 and f_{10}, better results are found in LPO without one-dimension search, while worse results are found for all other selected functions, which indicates that the brutal competition of individuals (one-dimension search) is another main factor that contributes to the performance of LPO.

Table 17 Mean best function values and standard deviations of function $f_8 - f_{13}$ with or without one-dimension search in each dimension. All results have been averaged over 50 runs

Function	With one-dimension search			Without one-dimension search		
	Mean	Std.	CPU	Mean	Std.	CPU
f_8	−12569.4866	1.3059×10^{-10}	41	−7343.0970	626.4107	53
f_9	9.8517×10^{-26}	2.3185×10^{-26}	14	0	0	16
f_{10}	3.6714×10^{-7}	3.0267×10^{-8}	18	8.8818×10^{-16}	0	17
f_{11}	5.9388×10^{-12}	1.6813×10^{-12}	26	1.3777	0.9129	68
f_{12}	1.7757×10^{-14}	1.2321×10^{-14}	51	0.0096	0.0080	67
f_{13}	2.4583×10^{-13}	1.7165×10^{-13}	50	2.9662	3.8398×10^{-5}	30

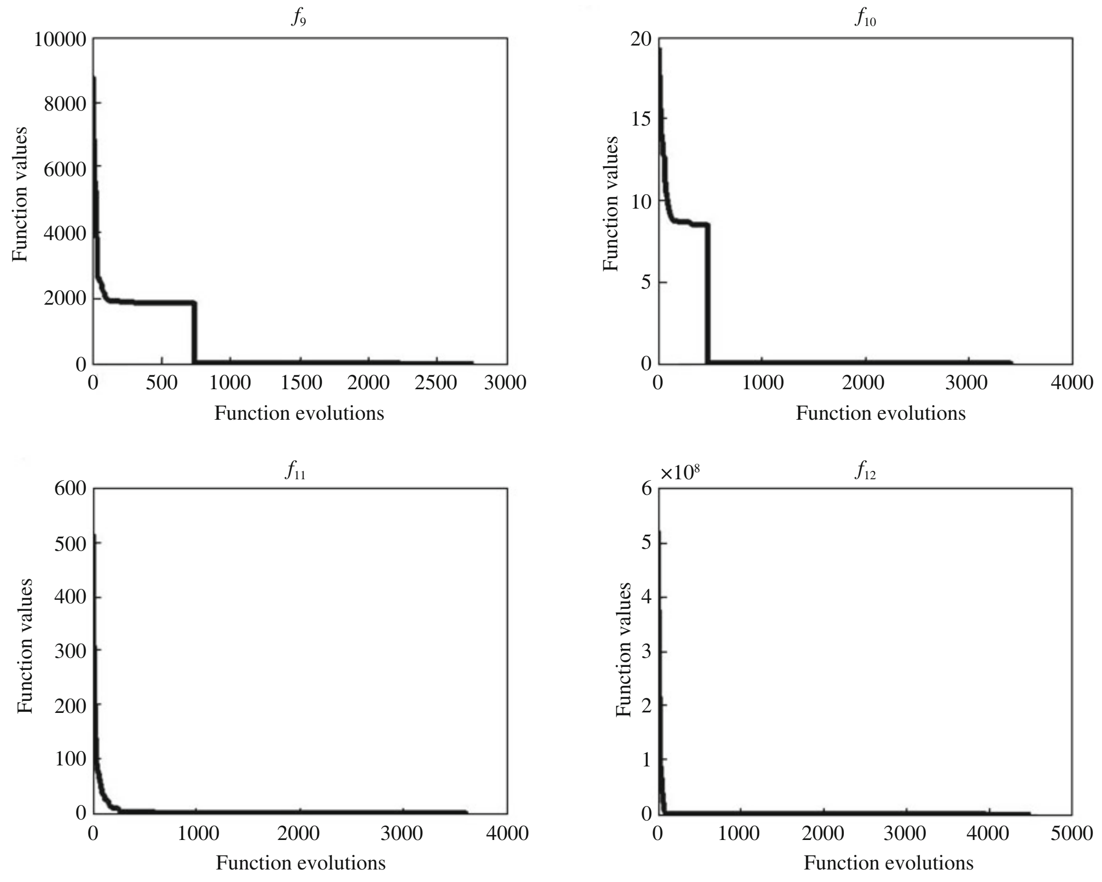

Figure 4 Convergence process of the algorithm on f_9–f_{12}.

4.4 Convergence process of the algorithm

The experiment results above present the performance of solving these problems, but it is also interesting to study the convergence process of the algorithm so that we can have a comprehensive understanding of algorithm capability. Since multimodal functions are regarded as the most challenging problems, here we select four multimodal functions f_9–f_{12} as the examples to show the convergence process of LPO (Figure 4). The graphs show that the algorithm converges fast, for f_9–f_{12}, LPO finds the best results after about 3000 function evolutions, much less than the maximal number of function evolutions for f_9–f_{12} (150000). And the results presented before show that optimization results are of high precision, which means that LPO is an algorithm with high precision and efficiency.

The graphs also show that the maximal number of function evolutions can be reduced to enhance the algorithm efficiency, since LPO already finds quite satisfaction results in few function evolutions. And in the graphs of f_9 and f_{10}, the effect of one-dimension search is obvious (a remarkable improvement under one function evolutions), which visually proves our conclusion that the brutal competition of individuals is another main factor that contributes to the performance of LPO.

5 Discussion

As mentioned in Section 1, there are several optimization algorithms inspired by animal behavior, and the most famous and successful one is ACO. Although both LPO and ACO draw inspiration from animal behavior, LPO is inspired by the lion pride evolution behavior, while ACO is inspired specifically by ant colonies' social foraging behavior: By laying pheromone trails, ants collectively establish the shortest path between their colony and feeding sources [2]. Another main difference is that ACO was proposed primarily for combinatorial optimization problems, whereas at present LPO is more applicable to continuous optimization problems.

PSO is another famous optimization algorithms inspired by animal behavior, and GSO is a newly emerging optimization algorithms inspired by animal behavior. Similar to LPO, PSO and GSO were proposed to solve continuous optimization problems. However, it is easy to find that they are different from LPO. PSO and GSO were inspired by social foraging behavior: PSO was originally developed from the models of coordinated animal motion such as Reynolds's Boids [43] and Heppner and Grenander's model [44]. Animal swarm behavior, mainly bird flocking and fish schooling, serves as the metaphor for the design of PSO. The GSO algorithm was inspired by general animal searching behavior. A generic social foraging model, e.g., PS model, was employed as the framework to derive GSO [2]. LPO is inspired by the lion pride evolution behavior, survival of the fittest is the main idea.

LPO shares more similarities with GAs than three other algorithms (PSO, GSO, and ACO). Both GAs and LPO were inspired by the evolution process of biology; and survival of the fittest is the main idea of both algorithms. However, there are also many differences between them. First and the fundamental one is that GAs was inspired by general biology evolution process, and survival of the fittest, with mating, and mutation as the main operators. LPO was inspired specifically by lion prides' evolution process. Lion prides' evolution process has its characteristic different from other biology evolution processes. The strongest males occupy nearly all mating resources, and if a new cohort of males is able to take over a pride, they will seek to kill young cubs of their predecessors. Thus, LPO more relies on the brutal competition of individuals than GAs, while GAs are closely related to the natural selection of individuals. Second, when compared with GAs, lion prides' evolution process has the characteristic of collective animal behavior. Each member plays an important role in the persistence of a lion pride and the evolution of the pride. The characteristic of LPO is more similar to serial computation, while that of GAs is more similar to parallel computation.

The comparison above indicates the differences and similarities between LPO, and GSO, ACO, PSO, and GAs. This may provide a possible explanation of why LPO could generate better results on most problems. Inspired by lion prides' evolution process, owing to the especial characteristic lion prides' evolution process, LPO is similar to a hybrid heuristic algorithm that combines the evolution strategy of GAs and collective animal behavior of SI. Thus, it is possible for LPO to better solve problems that are difficult for both GAs and SI.

6 Conclusions

We have proposed a novel optimization algorithm LPO in this paper, which is based on lion prides' evolution process and group living theory. This algorithm is conceptually simple and easy to implement.

From the studies of the algorithm property, it is found that the performance is not sensitive to most algorithm parameters, which shows the robustness of the algorithm and that the parameters are not problem-dependent. Central tendency of the algorithm is not found. But it is found that the pride update strategy and brutal competition of individuals, which are proposed in this study, are two main factors that contribute to the performance of LPO. LPO can solve a variety of optimization problems (including large scale), which makes it particularly attractive for real world applications.

A set of 23 benchmark functions have been used to test LPO in comparison with GSO, GA, PSO, CEP, FEP, CES, and FES, respectively. For the unimodal functions, the results show that the LPO finds better average results than those generated by the other seven algorithms in most benchmark functions. For most of the multimodal benchmark functions (5 of 6) with many local minima, LPO is able to statistically find better average results than those generated by the other seven algorithms. The test results obtained from the multimodal benchmark functions with a few local minima shows that LPO also outperforms the other seven algorithms. We also evaluated the LPO on a set of multimodal functions in 300 dimensions. In all these cases, the LPO generates better averaged results on all benchmark functions than those generated by GSO, GA, PSO, EP and ES.

Acknowledgements

This work was supported by National Natural Science Foundation of China (Grant No. 50875151). The authors thank Bu Hao (Southeast University), Yao Wei (Ohio State University Columbus), and Chen YiFeng (Tsinghua University) for their useful suggestion on the organization of this paper.

References

1 Bonabeau E, Dorigo M, Theraulaz G. Inspiration for optimization from social insect behaviour. Nature, 2000, 406: 39–42
2 He S, Wu Q H, Saunders J R. Group search optimizer: an optimization algorithm inspired by animal searching behavior. IEEE Trans Evol Comput, 2009, 13: 973–990
3 Barnard C J, Sibly R M. Producers and scroungers: A general model and its application to captive flocks of house sparrows. Anim Behav, 1981, 29: 543–550
4 Holland J H. Adaptation in natural and artificial systems. Ann Arbor: University of Michigan Press, 1975
5 Shiu Y Y, Chi K C. A genetic algorithm that adaptively mutates and never revisits. IEEE Trans Evol Comput, 2009, 13: 454–472
6 Eiben A E, Smith J E. Introduction to Evolutionary Computing. New York: Springer, 2003
7 Yao X, Liu Y, Liu G. Evolutionary programming made faster. IEEE Trans Evol Comput, 1999, 3: 82–102
8 Eiben A E, Hinterding R, Michalewicz Z. Parameter control in evolutionary algorithms. IEEE Trans Evol Comput, 1999, 3: 124–141
9 Mauldin M L. Maintaining diversity in genetic search. In: Proceedings of National Conference on Artificial Intelligence, Austin, USA, 1984. 247–250
10 Davis L. Handbook of Genetic Algorithms. New York: Van Nostrand Reinhold, 1991
11 Friedrich T, Hebbinghaus N, Neumann F. Rigorous analyses of simple diversity mechanisms. In: Proceedings of Genetic Evolutionary Computation Conference, London, UK, 2007. 1219–1225
12 Goldberg D E, Richardson J. Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of International Conference on Genetic Algorithms, Cambridge, USA, 1987. 41–49
13 Alba E, Tomassini M. Parallelism and evolutionary algorithms. IEEE Trans Evol Comput, 2002, 6: 443–462
14 Xin B, Chen J, Peng Z H, et al. An adaptive hybrid optimizer based on particle swarm and differential evolution for global optimization. Sci China Inf Sci, 2010, 53: 980–989
15 Tassing R, Guo L, Liu J, et al. Gene sorting in differential evolution with cross-generation mutation. Sci China Inf Sci, 2011, 54: 268–278
16 David M, Manu D B, Raf H, et al. Classification with ant colony optimization. IEEE Trans Evol Comput, 2007, 11: 651–665
17 AlRashidi M R, El-Hawary M E. A survey of particle swarm optimization applications in electric power systems. IEEE Trans Evol Comput, 2009, 13: 913–918
18 Valle Y D, Venayagamoorthy G K, Mohagheghi S, et al. Particle swarm optimization: Basic concepts, variants and applications in power systems. IEEE Trans Evol Comput, 2008, 12: 171–195
19 Linhares A. Synthesizing a predatory search strategy for VLSI layouts. IEEE Trans Evol Comput, 1999, 3: 147–152
20 Ray T, Liew K M. Society and civilization: An optimization algorithm based on the simulation of social behavior. IEEE Trans Evol Comput, 2003, 7: 386–396
21 Laumanns M, Rudolph G, Schwefel H P. A spatial predator-prey approach to multiobjective optimization: A prelim-

inary study. In: Proceedings of Parallel Problem Solving From Nature-PPSN V, Amsterdam, the Netherlands, 1998. 1498: 241–249

22 Passino K M. Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Syst Mag, 2002, 22: 52–67

23 Dasgupta S, Das S, Abraham A, et al. Adaptive computational chemotaxis in bacterial foraging optimization: an analysis. IEEE Trans Evol Comput, 2009, 13: 919–941

24 Muller S D, Marchetto J, Airaghi S, et al. Optimization based on bacterial chemotaxis. IEEE Trans Evol Comput, 2002, 6: 16–29

25 McComb K, Packer C, Pusey A. Roaring and numerical assessment in contests between groups of female lions (Panthera leo). Anim Behav, 1994, 47: 379–387

26 Parker C. Infanticide is no fallacy. Am Anthropol, 2000, 102: 829–831

27 Bygott J D, Bertram B C R, Hanby J P. Male lions in large coalitions gains reproductive advantages. Nature, 1979, 282: 839–841

28 Yao X, Liu Y. Fast evolution strategies. In: Proceedings of Evolutionary Programming VI. Berlin: Springer-Verlag, 1997. 151–161

29 Liang J J, Qin A K, Suganthan P N, et al. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans Evol Comput, 2006, 10: 281–295

30 Pusey A E, Packer C. The evolution of sex biased dispersal in lions. Behaviour, 1987, 101: 275–310

31 Mosser A, Packer C. Group territoriality and the benefits of sociality in the African lion, Panthera leo. Anim Behav, 2009, 78: 359–370

32 Kissui B M, Mosser A, Packer C. Persistence and local extinction of lion prides in the Ngorongoro Crater, Tanzania. Popul Ecol, 2010, 52: 103–111

33 Wolpert D H, Macready W G. No free lunch theorems for search. IEEE Trans Evol Comput, 1997, 1: 67–82

34 Töm A, Zilinskas A. Global Optimization. LNCS, vol. 350. Berlin: Springer-Verlag, 1989

35 Fogel L J, Owens A J, Walsh M J. Artificial intelligence through a simulation of evolution. In: Maxfield M, Callahan A, Fogel L J, eds. Biophysics and Cybernetic Systems: Proceedings of the Second Cybernetic Sciences Symposium. Washington: Spartan Books, 1965. 131–155

36 Fogel D B. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. New York: IEEE Press, 1995

37 Schwefel H P. Evolution and Optimum Seeking. New York: Wiley, 1995

38 Kennedy J, Eberhart R C. Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, vol. 4. Piscataway: IEEE Press, 1995. 1942–1948

39 Yao X, Liu Y. Scaling up evolutionary programming algorithms. In: Proceedings of the 7th Annual Conference on Evolutionary Programming, LNCS. Berlin: Springer-Verlag, 1998. 103–112

40 Liu Y, Yao X, Zhao Q, et al. Scaling up fast evolutionary programming with cooperative coevolution. In: Proceedings of the 2001 Congress on Evolutionary Computation. Piscataway: IEEE Press, 2001. 1101–1108

41 Lasserre J B. Global optimization with polynomials and the problem of moments. SIAM J Optimiz, 2001, 11: 796–817

42 Barhen J, Protopopescu V, Reister D. Trust: A deterministic algorithm for global optimization. Science, 1997, 276: 1094–1097

43 Reynolds C W. Flocks, herds and schools: A distributed behavioral model. Comput Graph, 1987, 21: 25–34

44 Heppner F, Grenander U. A stochastic nonlinear model for coordinated bird flocks. In: The Ubiquity of Chaos. Washington: AAAS Publications, 1990