A probabilistic cooperative–competitive hierarchical model for global optimization

K.S. Leung, I. King *, Y.B. Wong

Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong

Abstract

Stochastic searching methods have been applied widely to areas such as continuous and combinatorial optimization problems in a number of disciplines. Many existing methods solve these problems by navigating on the surface of the possibly rugged landscape. This kind of navigation is not very effective because the property of the landscape at different resolutions can be very different. Time spent at the beginning of the search on the detailed part of the landscape is often useless. Appropriate searching strategies should be adopted at different resolutions. In this paper, we propose a new probabilistic searching model for global optimization. The main contributions of the model are (1) to provide a basis for resolution control and smoothing of search space and (2) to introduce continuous memory into stochastic search. The basis of resolution control is achieved by dividing the search space into a finite number of n-dimensional partitions structurally. The number of partitions governs the resolution of the search space. The more the partitions, the finer is the search space and the more detailed and rugged is the landscape. The benefits are twofold. First, the rugged landscape problem can be smoothed, because the ruggedness is a matter of the number of partitions. Hence, the difficulty in search due to the ruggedness of the landscape can be controlled. Second, it provides a basis to implement algorithms that may change the ‘view’ of the landscape during the search process because we can dynamically divide the search space accord-

* Corresponding author.
E-mail address: king@cse.cuhk.edu.hk (I. King).

0096-3003/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2005.08.045
ingly. Another important feature that we use is *continuous memory*. Throughout the search process, searching experience is continuously accumulated in order to shape the global picture of the search space guiding the future searching direction.

We present results on the algorithm performance in handling numerical function optimization. The empirical results show that our new model is comparable to, and in many cases performs better than, that of the other advanced methods in terms of solution quality and computation required.

© 2005 Elsevier Inc. All rights reserved.

**Keywords:** Global optimization; Iterative probabilistic cooperative–competitive hierarchical search; Search space smoothing; Adaptive remembrance scheme; Meta-heuristic

1. Introduction

1.1. Motivation

Global optimization approaches under the category of stochastic search methods and heuristic search methods have been widely applied to many continuous and combinatorial optimization problems in different disciplines. To name a few, simulated annealing (SA) [25,26] has been applied to facility layout problem [33] and flexible object placement problem [1,2]. Evolutionary algorithms (EAs) [7,14,19], have been applied to routing [27], VLSI (Very Large Scale Integration) design [44], optimization [36] and combinatorial optimization. Genetic programming (GP) [28] has been applied to various computer program design [53]. Tabu search [15] has been applied to time-tableing [12], scheduling [10,32] and telecommunication routing [54]. Population-based incremental learning [6] has been applied to real-time tactical reasoning [47]. Multistart greedy descent strategies (MGDs) [20,40,49] have been applied to VLSI netlist bipartitioning [11], traveling salesmen problem [46] and scheduling [5].

Algorithms of these approaches share several common characteristics:

1. **Landscape without different resolutions**

   For SA, PBIL (Population-based Incremental Learning), cGA (Compact Genetic Algorithm) and MGDs, these algorithms use a single search space at the highest resolution during the search. This type of algorithm is inefficient and sometimes inflexible enough for problems with rugged landscapes, since the search space of these kind of problems at the highest resolution is typically too complex to be searched effectively. This is especially true when there is no analytical information about the solution. Meta-heuristic which can be used as local search operator in global optimization such as Variable Neighborhood Search (VNS) [34,18] works with all neighborhoods acting on the same degree of resolution too.
2. Search without memorization

Typically these algorithms do not use information accumulated during the search. Often, past information may help to guide the search in a more focused and efficient manner. Searching with memory is not entirely new. Tabu search which is considered as a meta-strategy for guiding other search heuristics keeps a tabu list of previous moves in order to avoid revisiting the same move. However, first the tabu list is limited in size while the search space is huge. Second the tabu list is used to avoid revisiting instead of using it to shape the global picture of the search space.

There are a number of works on evolutionary optimization to reduce the computation cost on evaluating fitness functions [4,24,31,39,43]. A major feature of these algorithms is incorporating approximation into fitness function evaluations. The approximation techniques include quadratic interpolation, least square estimation, Kriging model, etc. Besides reducing computational expenses, these techniques can smooth the rough search space. However, these approximation techniques are not directly related to the fitness functions to be solved, i.e., no matter what the fitness functions are, the same approximation model is applied. It would result in variations of performance on different fitness functions [30].

In view of all of the above, we present an alternative approach for global optimization.

1.2. Paper organization

Section 2 describes the essential idea of the probabilistic cooperative–competitive hierarchical model. Section 3 is the formulation of the model. Section 4 shows the experimental results to illustrate the behavior and performance of the model. Lastly, we make a few final remarks to summarize our work in Section 6.

2. Search space partitioning

Suppose we have a balanced binary hierarchy of \( l \) levels as shown in Fig. 1. If we need to make a decision on the branch to traverse next, we need to make \( l \) such decisions. Since a branch of the hierarchy leads to a unique non-overlapping sub-hierarchy below it, after making a decision on the selected branch to traverse, in principle we need only to consider the corresponding sub-hierarchy in the next decision.

It is clear that the size of the hierarchy we face diminishes with the decision made towards the bottom of the search space tree.

Viewing the hierarchy in another way, if we cut the hierarchy into two halves longitudinally at node level \([l/2]\) as shown in Fig. 2, the number of leaf
nodes faced by all sub-hierarchies at the upper half are reduced to $2^{l/2}$. Those in the lower half are, however, kept unchanged as mentioned before. In general, if we cut the hierarchy successively at each node level in a top-down manner, the total number of ‘leaf nodes’ faced is $2l$. Hence, the apparent size of the hierarchy can be reduced drastically.

The formation of such a hierarchy defines $l + 1$ number of resolution levels of the solution landscape. Node levels higher up in the hierarchy represent the coarser landscape, revealing the general macroscopic view; while node levels that are lower in the hierarchy represent the finer landscape, revealing more details. This resolution hierarchy allows an algorithm designed to concentrate on searching at the lower resolution, which is easier, locating the promising area first and to drive into the precise optimum later at the higher resolution when the search converges.

Given an $n$-dimensional continuous real-valued function, a typical function optimization can be expressed as follows:

$$F : X \rightarrow \mathbb{R} \quad \text{where } X \subseteq \mathbb{R}^n \quad \text{and} \quad x_{\text{smallest}} \leq X \leq x_{\text{largest}}.$$  

(1)
To optimize it, we need to find $x^* \in X$ such that $F(x^*)$ is maximized (or minimized).

To solve the above optimization problem, one may quantize the search space into equal partitions. The size of these partitions is related to the solution precision, i.e., the smaller the partition size, the finer the precision.

**Definition 1 (Basic partitioning).** Given a search space of $n$-dimensions, it contains $V^n$ number of equal partitions, where $V$ equals $2^l$. Each partition is of $n$-dimension.

For simplicity, we use binary partitioning, i.e., partition the search space into two equal halves recursively until $V$ number of partitions are created. It is not necessary to use binary partitioning scheme. In fact, any order will work as long as it is applicable to the problem to be solved. By creating this sample space with $V^n$ partitions, the optimization problem can then be modeled as a searching and approximation problem with $V^n$ number of choices in the $n$-dimensional space. Without loss of generality, we consider the one-dimensional case first.

**Definition 2 (Binary number labeling scheme).** Denoting $S$ as the set of all binary strings of length $l$ in the form of $b_{l-1} b_{l-2} \cdots b_0$, where $b_i \in \{0, 1\}$, we can label the partitions of the sample space of the one-dimensional function by assigning consecutive binary strings from 0 to $V - 1$ to consecutive partitions as illustrated in Fig. 3.

For instance, if $l$ equals 3, the partitions are represented sequentially as 000, 001, 010, ..., 110, and 111 in an increasing $x$ direction. Based on this labeling scheme, note that the one-dimensional search space is not only divided into $V$ partitions, but also a hierarchy of partitions with each bit demarcating the partition inherited from the immediate more significant bit into two halves.

![Fig. 3. Labeling of partitions: partitioning.](image-url)
For functions of $n$-dimension, we apply the same labeling scheme to each of the variables in $x$; hence, there are $n$ numbers separating the binary hierarchies. The optimization problem would then become $n$ simultaneous series of $l$ selections.

To locate the optimal solution, we adopt a probabilistic search. In this search, we give scores to the states of each bit $b_i$. Since we are considering a binary system, two scores are assigned, one to each state, indicating how well the states have performed in that bit position.

The problem that we are trying to solve is to find $x^* \in X$ where $X \subseteq \mathbb{R}^n$ such that

$$\forall x \in X \bullet \begin{cases} F(x^*) \geq F(x) & \text{if Maximization,} \\ F(x^*) \leq F(x) & \text{if Minimization.} \end{cases}$$

(2)

The problem can be transformed into finding an optimal vector of binary strings $s^* \in S^n$ to where $x^*$ belongs probabilistically:

$$\max \text{Prob}(\text{select } s^*) = \max \prod_{m=0}^{n-1} \text{Prob}(\text{select } s^*_m)$$

$$= \max \prod_{m=0}^{n-1} \prod_{i=0}^{l-1} \text{Prob}(\text{select } b^*_m,i),$$

(3)

where $s^*_m \in S$ is the $m$th component (the $m$th dimension) in vector $s^*$ and $b^*_m,i$ is the $i$th significant bit of binary string $s^*_m$.

It can then be reformulated as finding $b^*_m,i$ such that for $0 \leq m < n$ and $0 \leq i < l$,
\[ b_{m,i}^* = \arg \max_k \{a_{m,k} : k = 2(l - 1 - i) + b_{m,i}\}. \]  

(4)

where \( l \) is the length of bit string, \( i \) is the bit position of the bit string, \( k \) is the position of score vector, and \( a_{m,k} \) is the \( k \)th element of the score vector for the \( m \)th dimension, i.e., \( \text{Prob}(\text{select } b_{m,i}^*) \).

3. The model

To solve the problem formulated in the last section, we present in this section an iterative algorithm based on an information processing cycle characterized by a population of homogeneous searching agents and a searching environment (cf. Algorithm 1). We will describe the model progressively from: (1) the basic pBHS (probabilistic binary hierarchical search), to (2) pBHS with cooperation (pcBHS), and finally (3) pcBHS with competition (pccBHS).

3.1. Local searching agents

Each agent is designed to generate probabilistically in each time step \( n \) number of binary strings through \( n \) sequences of bit-value selection. We treat the set of scores \( a_{m,k}(t) \in [0.0, 1.0] \) at time \( t \) stated in Eq. (4) as our global information accumulated up to time \( t \). For each function variable \( x_m \), we define a vector

\[ A_m(t) = [a_{m,0}(t) \ a_{m,1}(t) \ a_{m,2}(t) \ \cdots \ a_{m,2l-1}(t)] \]

(5)

composed of \( 2l \) number of \( a_{m,k}(t) \) (two consecutive \( a_{m,k}(t) \) for one bit in a binary string of length \( l \)). For an \( n \)-dimensional problem, the entire set of scores would be

\[ A(t) = [A_0(t) \ A_1(t) \ \cdots \ A_{n-1}(t)]^T. \]  

(6)

In order to make the selection possible, a correspondence is drawn between \( A_m(t) \) and our binary string \( s_m \). Every pair of two consecutive \( a_{m,k}(t) \) is used to represent a single bit. For instance, elements \( a_{m,0}(t) \) and \( a_{m,1}(t) \) correspond to the most-significant bit \( b_{l-1} \), \( a_{m,2}(t) \) and \( a_{m,3}(t) \) correspond to the second-most-significant bit \( b_{l-2} \), and so on. For each such pair of elements, we dedicate the former one as the score for \( b_i = 0 \) and the latter one as the score for \( b_i = 1 \). For instance, \( a_{m,0}(t) \) is the score of 0 in bit \( b_{l-1} \), and \( a_{m,1} \) is the score of 1 in bit \( b_{l-1} \). Fig. 5 shows the correspondence of a binary string and \( A_m(t) \). In fact, it is not necessary that \( A_m(t) \) and the correspondence be defined as above. Different applications may have different definitions.

**Definition 3** (Bit-value selection probability). The probability of selecting a bit-value at the \( i \)th bit \( b_{m,i} \) of the \( m \)th string \( s_m \) is defined as follows:
\[
\text{Prob}(b_{m,i} = \zeta) = \begin{cases} 
    a_{m,k}(t), & \zeta = 0, \\
    1 - a_{m,k}(t), & \zeta = 1,
\end{cases}
\tag{7}
\]

where \( k = 2(l - 1 - i) + b_{m,i} \).

As shown in Eq. (7), the selection of the bit-value depends on the respective global information complying with Eqs. (3) and (4). The larger the \( a_{m,k} \) value, the higher the chance that the corresponding bit-value is selected.

After generating the binary strings \( s_m, 0 \leq m < n \) for all function variables \( x_m \), we have a designated \( n \)-dimensional partition. Since the ultimate goal is to optimize the original function stated in Eq. (2), we need a function value \( x \) from the partition for evaluation. This value is chosen according to

\[
x_m = \frac{S_m}{V} (x_{m,\text{largest}} - x_{m,\text{smallest}}) + x_{m,\text{smallest}},
\tag{8}
\]

i.e., the minimum \( x \) in the region \( s_m \). Now we have a way of evaluating the partition by evaluating the binary string instead.

### 3.2. Global environment

Given a reliable global information \( A^* \), the searching agents described in the above section is able to find \( s^* \) with probability approaching 1.0 fulfilling Eq. (3), i.e., \( \text{Prob(select } s^*) \approx 1 \). The question is how to make \( A^* \) reliable. We will answer this in the next section.

Every set of binary strings \( s_m, 0 \leq m < n \) generated will be evaluated to give a function value \( F(x_m) \), which is the raw fitness of the binary strings. Two sets of vectors of length \( l \) for the raw fitness values of both states gained by each binary string are defined. We denote \( u_{m,j} = [u_{m,j,0} \ u_{m,j,1} \ldots \ u_{m,j,l-1}] \) as a vector indicating the raw fitness of the bits with bit-values equal to 0 for the \( j \)th binary string \( s_{m,j} \) of the \( m \)th dimension in the population and
\(w_{m,j} = [w_{m,j,0} \ w_{m,j,1} \ \ldots \ w_{m,j,l-1}]\) as the vector indicating the raw fitness of bits with bit-values equal to 1.

For the \(j\)th binary string \(s_{m,j}\) of \(m\)th dimension, and \(0 \leq i < l\), the fitness assignment to the states of each component is defined as

\[
\begin{align*}
    u_{m,j,i} &= f_j \quad \text{and} \quad w_{m,j} = 0 \quad \text{if} \quad b_{m,l-1-i} = 0, \\
    u_{m,j,i} &= 0 \quad \text{and} \quad w_{m,i} = f_j \quad \text{if} \quad b_{m,l-1-i} = 1,
\end{align*}
\]

(9)

where \(f_j\) is a normalized raw fitness of the \(x_j\) in the population. This normalized raw fitness is defined as

\[
f_j = \frac{F(x_j) - F_{\text{min}}}{F_{\text{max}} - F_{\text{min}}},
\]

where \(F_{\text{max}} = \max F(x_i)\) and \(F_{\text{min}} = \min F(x_i)\), for \(0 \leq i < N\). The purpose of the normalization is to maintain sufficient convergence force when the raw fitness of the samples is getting closer to each other on flat landscape or at the sharp peak.

**Definition 4 (Component fitness).** For a population of size \(N\), we have two sets of \(N\) raw fitness vectors \(u_m\) and \(w_m\). Summation of all the same components of the \(N\) vectors of the respective sets gives a component fitness for the respective states. Denoting \(u_{j,m,i}\) and \(w_{j,m,i}\) as the raw fitness values for states 0 and 1 in the \((l-1-i)\)th bit gained from evaluating the \(j\)th binary string in the population for variable \(x_m\) respectively, the component fitness values for both states of the \((l-1-i)\)th bit \(b_{m,j,(l-1-i)}\) resulted from the population are

\[
\begin{align*}
    U_{m,i} &= \frac{\sum_{j=0}^{N-1} u_{m,j,i}}{|Q_{m,i,0}|}, \\
    W_{m,i} &= \frac{\sum_{j=0}^{N-1} w_{m,j,i}}{|Q_{m,i,1}|},
\end{align*}
\]

(11)

where \(Q_{m,i,\zeta} = \{j \in \{0,1,\ldots,N-1\}: b_{m,j,l-1-i} = \zeta\}\) and \(\zeta = \{0,1\}\), while \(U_m\) and \(W_m\) are vectors with \(l\) number of vector components:

\[
\begin{align*}
    U_m &= [U_{m,0} \ U_{m,1} \ \ldots \ U_{m,l-1}], \\
    W_m &= [W_{m,0} \ W_{m,1} \ \ldots \ W_{m,l-1}].
\end{align*}
\]

(12) (13)

Vectors \(U_m\) and \(W_m\) are normalized such that \(U_{m,i} + W_{m,i} = 1, \ 0 \leq i < l\).

Putting \(U_m\) and \(W_m\) together, we obtain a vector of combined component fitness with the same structure as \(A_m\)

\[
H_m = [U_{m,0}W_{m,0} \ U_{m,1}W_{m,1} \ \ldots \ U_{m,l-1}W_{m,l-1}].
\]

(14)

These component fitness values are used by the searching agents to produce next generation of populations (binary strings). Since this component fitness is only calculated using the last generation information, the population in the next generation is produced using the immediate past memory. In order to use all past memory, we need to accumulate all past component fitness val-
ues. The past component fitness values for the $m$th function variable are accumulated as follows.

**Definition 5** (Accumulation of past searching experience). Given $h_{m,k}(t-1)$ as the $k$th component of $H_m$ at time $t-1$, $0 \leq i < l$, we define the accumulation of past searching experience as

$$a_{m,k}(t) = \beta_{m,i}(t-1) \cdot a_{m,k}(t-1) + (1 - \beta_{m,i}(t-1)) \cdot h_{m,k}(t-1),$$

where $k = 2(l-1-i)$ for state 0, and $k = 2(l-1-i) + 1$ for state 1.

In practical applications, we keep every antagonistic pair inside $A_m(t)$ normalized: $a_{m,k}(t) + a_{m,k+1}(t) = 1$.

The newly introduced quantity $\beta_{m,i}(t)$ is called remembrance. It determines the fraction of the past collected information $a_{m,k}(t-1)$ to be retained in generation $t$. Although the past collected information is treated the same way in each iteration, effectively the older the collected information, the lesser the amount remained and hence the lesser the influence.

It is defined so that different bits can have different remembrance values. There are two reasons why different bits should have different remembrances:

1. **Convergence from more significant bits towards less significant bits**
   Intuitively, after a series of searching (sampling), the more significant bits should have collected enough information in the exploration phase than the less significant bits, because the more significant the bits the large the partition of the search space they represent and hence the coarser the search landscape they represents. Given enough sampling, smaller remembrance values (effectively discourage exploration and encourage exploitation) for the more significant bits can be used to speed up the convergence of the more significant bits.

2. **Search space reduction**
   The hierarchical structure has the advantage of search space reduction. Briefly, suppose that sufficient information is collected through sampling at a certain level of the hierarchy all the way up to the top, we can ignore these levels in future sampling which effectively reduces the search space. For instance, if the most-significant bit $b_{m,t-1}$ collected enough information, either $a_{m,0}(t)$ or $a_{m,1}(t)$ will have very high value. Say if $a_{m,1}(t)$ has a higher value, it is highly probable that the right partition contains the global optimum. Searching should then be concentrated on that region. In other words, the size of the search space is reduced by half, suggesting a smaller remembrance value be used to accelerate the convergence.

Due to the abovementioned reasons, we devised an *adaptive remembrance scheme* to speedup the convergence.
**Definition 6** (Adaptive remembrance scheme). Let \( \tau \) denote a convergence threshold. A score \( a_{m,k}(t) \) converges when it is greater than \( \tau \). Let \( \beta \) denote the minimum allowed remembrance. Suppose the \( r \)th bit \( b_{m,r} \) of binary string \( s_m \) for function variable \( x_m \) is the first bit encountered starting from the most significant side that satisfies the following:

\[
|0.5 - a_{m,2(l-1-r)}(t)| > \tau \vee |0.5 - a_{m,2(l-r)}(t)| < \tau.
\]

Then the remembrance value used in each bit of \( s_m \) is set according to:

\[
\beta_{m,i}(t) = \begin{cases} 
\beta & l > i \geq r, \\
\frac{r-i+\beta}{r+1} & r > i \geq 0.
\end{cases}
\]

This scheme, basically, keeps the remembrance for the converged bits (\( b_{l-1} \) to \( b_{r+1} \)) constant at \( \beta \), while interpolates the rest from \( \beta \) to \( (r+\beta)/(r+1) \). Fig. 6 shows the remembrance settings at different stages of convergence.

### 3.3. pcBHS: pBHS with cooperation

For many optimization problems, especially when there is no analytical information about the problem, high-dimensionality sometimes poses a challenge to many optimization algorithms, because of the exponential scale-up of the size of the search space. To handle the high-dimensional problem, we introduce pcBHS (Probabilistic Cooperative Binary Hierarchical Search) incorporating cooperation. In addition to the cooperation among searching...
agents in the basic model described before, the model incorporates cooperation among the dimensions of the problem [37].

In the basic pBHS model, a population is defined as a group of sample points consisting of samples from all sub-spaces. In the following, sample point is referred to as a complete solution, while a sample from a sub-space is referred to as a solution fragment. For instance, a complete solution for a three-dimensional function $F(x)$ is a vector $[x_0, x_1, x_2]$, which consists of three solution fragments: $x_0$, $x_1$, and $x_2$.

In pcBHS, decoupling is taken such that each sub-space exists as its own and a single population in the pBHS model becomes $n$ subpopulations here. The size of each subpopulation is still kept at $N$ in order to maintain the diversity of solution fragments as in pBHS. The situation is illustrated in Fig. 7. The shaded region enclosed by two dotted lines indicates a single complete solution in the basic pBHS model. In pcBHS, all solution fragments in a subpopulation are not tied with any solution fragment in other subpopulations. What we have are $n$ sets of $N$ solution fragments.

Before describing how to combine solution fragments, the issue on fitness measurement should be addressed first. The ordinary fitness measurement used in the pBHS becomes inappropriate in the cooperative model. Raw fitness is meaningful only when a single complete solution exists. After decoupling, fragments representing the problem sub-spaces are created. Their fitness values are undefined. Cooperative fitness as defined in [37] is employed to evaluate the solution fragments. Given $n$ arbitrary solution fragments $\{x_0, x_1, \ldots, x_{n-1}\}$ from each subpopulation, each of their raw cooperative fitness equals $F(x)$ where $x = [x_0, x_1, \ldots, x_{n-1}]$. Suppose that the same set of solution fragments is given with $x_{n-1}$ replaced by $x_{n-1}'$, their raw cooperative fitness become $F(x')$ where $x' = [x_0, x_1, \ldots, x_{n-1}']$.

![Fig. 7. Decoupling of dimensions in a population for a three-dimensional problem.](image-url)
The cooperative pcBHS model differs from the basic pBHS model in three aspects: (1) fitness evaluation, (2) fitness scaling, and (3) elitism. We now describe them briefly.

1. **Fitness measurement and fitness scaling**
   As discussed in the previous sections, raw fitness is replaced by cooperative fitness owing to the decoupling of solution fragments. Suppose that there is a global elite \( x^e = [x_0^e, x_1^e, \ldots, x_{n-1}^e] \), the cooperative fitness of each solution fragment \( x_{m,j} \) in each subpopulation \( m \) is defined as \( cF(x_{m,j}, x^e) \). Function \( cF \) is simply the objective function \( F \) applied to a complete solution formed by replacing the \( m \)th element in \( x^e \) by \( x_{m,j} \). Algorithm 2 shows how it is implemented. Under this scheme, there are \( n \times N \) complete solutions centered around the \( x^e \) that was formed. These raw cooperative fitness variables, \( cF \), of each solution fragment are scaled within their subpopulations only. Denoting \( cf \) as the scaled cooperative fitness, the \( cf \) of the \( j \)th individual in the \( m \) subpopulation is defined as
   \[
   cf(x_{m,j}, x^e) = \frac{cF(x_{m,j}, x^e) - cF_{\text{min}}^m}{cF_{\text{max}}^m - cF_{\text{min}}^m},
   \]
   where \( cF_{\text{max}}^m = \max \{F(x^e), \max_{0 \leq j \leq n-1} cF(x_{m,j}, x^e) \} \) and \( cF_{\text{min}}^m = \min \{F(x^e), \min_{0 \leq j \leq n-1} cF(x_{m,j}, x^e) \} \). In Eqs. (9) and (10), it is \( f_j \) that is fed back into the system. We now use the scaled cooperative fitness \( cf \). Given a binary string \( s_m \) of the \( m \)th dimension, and \( 0 \leq i < l \), the component fitness for the \((l - 1 - i)\)th bit is determined as follows:
   \[
   \begin{align*}
   u_{m,i} &= cf(x_{m,j}, x^e) \quad \text{and} \quad w_{m,i} = 0 \quad \text{if} \quad b_{m,l-1-i} = 0, \\
   u_{m,i} &= 0 \quad \text{and} \quad w_{m,i} = cf(x_{m,j}, x^e) \quad \text{if} \quad b_{m,l-1-i} = 1.
   \end{align*}
   \]

2. **Elitism**
   Under this model, the elitist strategy used in the basic pBHS model has to be modified. Since each subpopulation is individually responsible for a single unique dimension, elitism is applied separately to each subpopulation (see Algorithm 2) in each generation producing a set of new local elites \( \{x_0^{e}, x_1^{e}, \ldots, x_{n-1}^{e}\} \). The new global elite \( x^{e'} \) is selected from the solutions produced by the following three schemes:
   - **Scheme 1:** No-change
     The existing global elite \( x^e \) with raw fitness \( F(x^e) \).
   - **Scheme 2:** Local
     Local elite \( x_m^{e} \) in cooperation with the existing global elite \( x^e \). In this scheme, there are \( n \) elite candidates. We take the best of these candidates with the highest fitness as the representative of this scheme. Local elite \( x_m^{e} \) from the \( m \)th subpopulation replaces the \( m \)th solution fragment in \( x^e \) to produce an elite candidate. Since there are \( n \) subpopulations, there are \( n \) elite candidates.
• Scheme 3: Multiple
A complete solution formed by combining the existing global elite $x^e$ and those $x^e_m$ whose $cF$ is greater than $F(x^e)$:

$$\forall m, 0 \leq m < n \text{ s.t. } cF(x^e_m, x^e) \geq F(x^e).$$

(20)

Suppose that the set of local elites that satisfies this criterion is $\psi = \{x^e_1, x^e_2\}$, the complete solution would be $\{x^e_0, x^e_1, x^e_2, \ldots, x^e_n\}$ and its cooperative fitness is denoted as $cf(\psi, x^e)$.

The global elite is selected according to

$$F(x^{e\text{c}}) \geq \max\{F(x^e), cF(x^e_m, x^e), cF(x^e_m, x^e), cF(\psi, x^e)\}.$$  

(21)

The last choice (multiple) is used to lower the greediness of the simple no-change plus local scheme of CCGA-1 as illustrated in [37]. Although the replacement scheme is still a winner-take-all strategy, the multiple scheme allows multiple-subspace movement in one single step.

3.4. pccBHS: The cooperative–competitive model

The two basic design goals of the model described so far have two characteristics: (1) distribution of a population of individuals who search cooperatively for a single global optimum and (2) assuming no (or minimal) a priori analytical information about the problem is available. We understand that every optimization algorithm has implicit assumptions (or limitations) on the problems to be solved [52]. What we mean here is that we have no analytic information about the problem and simply treat the problem as a black box. However, this design would make both algorithms be deceived easily, (1) when the landscape of a problem has similar basin of attractions, and (2) the landscape of a problem to be solved provides misleading information. In this section, we extend the model by introducing redundancy and competition. The enhanced model, called pccBHS (Probabilistic cooperative–competitive binary hierarchical search), shares similarities with some sharing mechanisms, such as the crowd principle in [42]. However, our model avoids using niche radius [9]. Niche radius inherently limits the niching mechanisms to be applied to problems that require niches to be located at different resolution levels simultaneously. The drawback of the pccBHS model is that the number of niches to be occupied is bound by a prescribed number.

The main structural characteristic of the pccBHS model is the division of a whole population into a number of groups (subgroups) to provide redundancy. They gather their own set of global information. The one with the highest fitness is considered as the global solution. Throughout the search, these subgroups compete with each other for exclusive occupancy of territories. The aim of the
competition is to force them to search different areas by separating them in the n-dimensional space. The competition is achieved by generating a repulsive force when two subgroups come together in the n-dimensional space. The closer the two subgroups, the greater the repulsive force and the lower the fitness of each of the individuals in both subgroups. Once they are separated, the force diminishes. Effectively, pccBHS remodels the function landscape in such a way that the deceptive attractor is made hidden by another subgroup. What a subgroup faces is a unimodal or non-deceptive landscape. Fig. 8 illustrates the idea.

In the pcBHS model, cooperative fitness \( cf(x_{m,j}, x^c) \) is used to evaluate the quality of \( x_{m,j} \) as in Eq. (19). In the pccBHS model, we add a competition factor to the cooperative fitness giving a resultant fitness called Interaction Fitness. Given a single large population, we divide it into \( G \) numbers of subgroups \( g_r \), \( 0 \leq r < G \) of size \( \frac{N}{G} \). The pcBHS model described in Section 3.3 is a special case of the pccBHS model with \( G = 1 \). For the sake of clarity, the model is presented using only two subgroups \( (G = 2) \). In an n-dimensional space, two subgroups, \( g_1 \) and \( g_2 \), are said to be in competition and hence have repulsive force acting on each other when they are mutually overlapped in all dimensions. The more the overlapping, the larger the repulsive force and the lower the interaction fitness they should have. In order to calculate the Interaction Fitness, we need to calculate the repulsive force which is in turn dependent on two other metrics: (1) degree of overlapping and (2) proximity (see Algorithm 3).

**Definition 7** *(Degree of overlapping)*. Denote \( g_{1 \text{min}}^{m} \) and \( g_{1 \text{max}}^{m} \) as the minimum and the maximum of the \( m \)th dimension solution fragment of \( g_1 \) respectively, and \( g_{2 \text{min}}^{m} \) and \( g_{2 \text{max}}^{m} \) as the minimum and the maximum of the \( m \)th dimension solution fragment of \( g_2 \) respectively. Without loss of generality, we assume \( g_{1 \text{min}}^{m} \leq g_{2 \text{min}}^{m} \). Degree of overlapping \( D_m(g_1,g_2) \) between \( g_1 \) and \( g_2 \) on the same dimension \( m \) is defined as

\[
D_m(g_1,g_2) = \frac{O_m}{F_m},
\]  

(22)
where

\[
O_m = \begin{cases} 
0 & \text{if } g_{1m}^{\max} < g_{2m}^{\min} \text{ or } g_{2m}^{\max} < g_{1m}^{\min}, \\
g_{1m}^{\max} - g_{2m}^{\min} & \text{otherwise}
\end{cases}
\]  

(23)

is the Euclidean distance between two farthest points of the two subgroups in the overlapping region representing how much \(g_1\) and \(g_2\) are overlapped (the shaded region in Fig. 9), and

\[
F_m = g_{2m}^{\max} - g_{1m}^{\min}
\]  

(24)

is distance between two farthest apart individuals in \(g_1\) and \(g_2\).

There are three distinct cases:

1. *Disjoint* \(g_1\) and \(g_2\) are totally separated.

   \[
   D_m(g_1, g_2) = 0 \quad \text{when } O_m = 0.
   \]  

   (25)

2. *Enclosure* \(g_1\) is totally enclosed by \(g_2\) or vice versa

   \[
   D_m(g_1, g_2) = 1
   \]

   when

   \[
   g_{1m}^{\min} < g_{2m}^{\min} \land g_{1m}^{\max} > g_{2m}^{\max}.
   \]  

   (26)

3. *Overlapping* \(g_1\) and \(g_2\) are overlapping

   \[
   0 < D_m(g_1, g_2) < 1
   \]

   when

   \[
   g_{1m}^{\min} < g_{2m}^{\min} < g_{1m}^{\max} < g_{2m}^{\max}.
   \]  

   (27)

The quantity \(D_m\) serves two purposes: (1) decides whether the repulsion exists and (2) determines the level of force required if repulsion exists. However, it
does not reflect the fact that individuals farther away from the overlapping subgroup should receive less repulsive force. Thus, a *proximity value* is needed.

**Definition 8 (Proximity).** For the \( m \)th dimension, proximity value \( P_m(g_1, x_{m,j}) \) is defined as the normalized distance between \( x_{m,j} \) of \( g_2 \) and the elite \( x^e_m \) of \( g_1 \) as shown in Fig. 9

\[
P_m(g_1, x_{m,j}) = \frac{|x_{m,j} - x^e_m|}{F_m},
\]

where \( P_m(g, x_{m,j}) \in [0, 1] \).

The farther away an individual in a subgroup from the elite of its neighboring overlapped subgroup, the larger the proximity value it has.

**Definition 9 (Repulsive force).** The repulsive force \( R_m \in [0, 1] \) experienced by an individual \( x_{m,j} \) in \( g_2 \) due to the overlapping with \( g_1 \) is defined as

\[
R_m(g_1, x_{m,j}) = D_m(g_1, g_2) \times (1 - P_m(g_1, x_{m,j})).
\]

The larger the degree of overlapping between two subgroups and the closer the individual \( x_{m,j} \) with the neighboring subgroup, the larger the repulsive force is acting on it.

Finally, *interaction fitness* \( I_m(x_{m,j}, x^e) \) is defined to indicate how well an individual performs in the competition.

**Definition 10 (Interaction fitness).** For the \( m \)th dimension,

\[
I_m(x_{m,j}, x^e) = cf(x_{m,j}, x^e) \times (1 - R_m(g_1, x_{m,j})),
\]

where \( cf(x_{m,j}, x^e) \) is the cooperative fitness of \( x_{m,j} \) as in Eq. (19).

Instead of feeding back \( cf \) into the system, \( I_m \) should be used:

\[
\begin{align*}
I_m(x_{m,j}, x^e) = cf(x_{m,j}, x^e) \times (1 - R_m(g_1, x_{m,j})),
\end{align*}
\]

With both cooperative and competitive components, the enhanced model, \( pccBHS \), is summarized in Algorithm 4.

### 4. Experiments

We have carried out experiments testing the algorithms on a number typical function optimization problems. The purpose of the experiments is to show the capability of the algorithms, demonstrating the power of the various aspects of the algorithms, such as the cooperation and the competition features, and comparing the algorithm (pccBHS) with other well-known function optimization algorithms.
We have three tests: (1) basic, (2) high-dimensionality, and (3) deception. The first test illustrates the capability of pBHS on solving four typical functions. The second test illustrates the power of the pcBHS on handling high-dimensional problems. The final test illustrates the power of pccBHS on handling deceptive problems.

4.1. Basic

In this experiment, we tried four minimization problems: Rastrigin R2, Goldstein–Price GP2, Hartman H3 and Shekel S1 (see Appendix B). They are constrained optimization problems commonly used in testing global optimization algorithms. R2 is a two-dimensional Rastrigin’s function having 50 optima with one global optimum \( F_{R2}(x^*) = 2.000000 \) located at \( x = (0,0) \). GP2 is Goldstein–Price’s function \([17,23]\) having the global optimum \( F_{GP2}(x^*) = -3.000 \) at \( x = [0-1] \). H3 is the three-dimensional function in Hartman’s family \([49]\). S1 is the one-dimensional function in Shekel’s family \([41,49]\) having global optimum \( F_{S1}(x^*) = 14.5926520 \) located at \( x = 0.6858609 \) where \( m = 10 \).

Table 1 lists the parameters used in the experiment. Tables 2 and 3 show the percentage of trials and average iterations required to get the global optima of

Table 1
Experimental conditions for pBHS test

<table>
<thead>
<tr>
<th>Functions</th>
<th>( N )</th>
<th>( \beta )</th>
<th>( \mu )</th>
<th>( \tau )</th>
<th>( l )</th>
<th>No. of trials</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td>10, 50, 90</td>
<td>0.80, 0.85, 0.90, 0.95</td>
<td>1</td>
<td>0.4</td>
<td>16</td>
<td>100</td>
</tr>
<tr>
<td>GP2</td>
<td>10, 50, 90</td>
<td>0.80, 0.85, 0.90, 0.95</td>
<td>1</td>
<td>0.4</td>
<td>16</td>
<td>100</td>
</tr>
<tr>
<td>H2</td>
<td>10, 50, 90</td>
<td>0.80, 0.85, 0.90, 0.95</td>
<td>1</td>
<td>0.4</td>
<td>16</td>
<td>100</td>
</tr>
<tr>
<td>S1</td>
<td>10, 20, 30</td>
<td>0.93, 0.94, 0.95, 0.96</td>
<td>1</td>
<td>0.4</td>
<td>16</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 2
Percentage of trials for pBHS to reach the global optimum for S1, R2, GP2, and H3 (\( \beta \) is remembrance value, \( N \) is population size)

<table>
<thead>
<tr>
<th>( \beta )</th>
<th>S1</th>
<th>R2</th>
<th>GP2</th>
<th>H3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N = 10 )</td>
<td>( N = 20 )</td>
<td>( N = 30 )</td>
<td>( N = 10 )</td>
<td>( N = 50 )</td>
</tr>
<tr>
<td>0.96</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.95</td>
</tr>
<tr>
<td>0.95</td>
<td>99</td>
<td>100</td>
<td>100</td>
<td>0.90</td>
</tr>
<tr>
<td>0.94</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>0.85</td>
</tr>
<tr>
<td>0.93</td>
<td>98</td>
<td>100</td>
<td>100</td>
<td>0.80</td>
</tr>
</tbody>
</table>

| \( N = 10 \) | \( N = 50 \) | \( N = 90 \) | \( N = 10 \) | \( N = 50 \) | \( N = 90 \) |
| 0.95       | 100 | 100 | 100 | 0.95 | 100 | 98  | 100 |
| 0.90       | 100 | 100 | 100 | 0.90 | 95  | 96  | 98  |
| 0.85       | 99  | 100 | 100 | 0.85 | 83  | 92  | 98  |
| 0.80       | 99  | 100 | 100 | 0.80 | 74  | 90  | 95  |
the respective functions. To obtained the average, we performed 100 consecutive trials for each experimental conditions. Hence, we obtained 100% success rate (reaching the prescribed $f^+$) on $S_1$ and $GP_2$ functions under majority of our experimental conditions. Comparing the amount of computation required, our algorithm is in general comparable with many well-known algorithms (see Tables 3 and 4). While for $R_2$ and $H_3$ functions, the drop in success rates (compared with $S_1$ and $GP_2$) for some test conditions can be explained by their rugged landscape and the high-dimensionality. This indicates that the parameter $\beta$ can be adjusted to fit different landscape properties. The more rugged the landscape, the larger the $\beta$ and hence the more the exploration effect. If pre-knowledge is completely unavailable, one can use a large $\beta$ value. There are two advantages for using a large $\beta$ value in this case: (1) better for rugged landscape, and (2) facilitates exploration.

4.2. High-dimensionality

In this experiment, several testing functions in 4 families ($S_5$, $S_7$ and $S_{10}$ in Shekel family, $H_3$ and $H_6$ in Hartman family, $A_{30}$ and $A_{100}$ in Ackley family, $R_{20}$ and $R_{100}$ in Rastrigin family) with problem size up to 100 dimensions are tried (See Appendix B). These problems are commonly used in testing global optimization algorithms. Each family of the problems possesses characteristics quite different from each other. Table 5 lists the published results of two advanced GA techniques: Breeder genetic algorithm (BGA) [35] and Evolutionary algorithm with soft genetic operators (EASY) [50,51].

The result of the experiment is listed in Table 6. It shows that the performance of our algorithm is comparable with the two existing advanced GA
Table 4
Performance of several well-known algorithms on solving S1, R2, GP2 and H3

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Function evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S1</td>
</tr>
<tr>
<td>MS—multistart [40,49]</td>
<td>– –</td>
</tr>
<tr>
<td>CRS—controlled random search [38]</td>
<td>– –</td>
</tr>
<tr>
<td>SA—simulated annealing [13]</td>
<td>– –</td>
</tr>
<tr>
<td>SAasde—SA based on stochastic differential equations [3]</td>
<td>– –</td>
</tr>
<tr>
<td>HGA—hybrid genetic algorithm [21]</td>
<td>– –</td>
</tr>
<tr>
<td>ARS—adaptive simulated annealing [22]</td>
<td>– –</td>
</tr>
<tr>
<td>NP—new Price’s algorithm [8]</td>
<td>– –</td>
</tr>
<tr>
<td>PE—Perttunen’s method [45]</td>
<td>– –</td>
</tr>
<tr>
<td>GAβ—genetic algorithm</td>
<td>1185.9</td>
</tr>
</tbody>
</table>

Experiment conditions for the GA test

- Population size 30 50 30 90
- Mutation probability \( P_\mu \) \(1/\text{n}_l, \text{l} = 16\)
- Crossover probability \( P_x \) 1.0 (two-point)
- Fitness fitness scaling
- Selection 2-tournament
- Replacement Proportional
- Success rate 100%

–: No results reported.

a Number of iterations.
b Experiments carried by ourselves.

Table 5
Benchmark results of two GA variants—Breeder GA (BGA) and EA with soft genetic operators (EASY) on Rb2, S5, S7, S10, H3, H6, A30, A100, R20 and R100

<table>
<thead>
<tr>
<th>Problems</th>
<th>n</th>
<th>( f^\star )</th>
<th>#eval</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rb2—Rosenbrock</td>
<td>2</td>
<td>0.001</td>
<td>7700</td>
<td>†</td>
</tr>
<tr>
<td>S5—Shekel</td>
<td>4</td>
<td>9.9</td>
<td>5403</td>
<td>★</td>
</tr>
<tr>
<td>S7—Shekel</td>
<td>4</td>
<td>9.9</td>
<td>5386</td>
<td>★</td>
</tr>
<tr>
<td>S10—Shekel</td>
<td>4</td>
<td>9.9</td>
<td>5862</td>
<td>★</td>
</tr>
<tr>
<td>H3—Hartman</td>
<td>3</td>
<td>3.86</td>
<td>1014</td>
<td>★</td>
</tr>
<tr>
<td>A30—Ackley</td>
<td>30</td>
<td>0.001</td>
<td>13,997/19,420</td>
<td>†</td>
</tr>
<tr>
<td>A100—Ackley</td>
<td>100</td>
<td>0.001</td>
<td>57,628/53,860</td>
<td>†</td>
</tr>
<tr>
<td>R20—Rastrigin</td>
<td>20</td>
<td>0.9</td>
<td>6,098/3,608</td>
<td>†</td>
</tr>
<tr>
<td>R100—Rastrigin</td>
<td>100</td>
<td>0.9</td>
<td>45,118/25,040</td>
<td>†</td>
</tr>
</tbody>
</table>

★: A clustering technique: New Price’s algorithm [8].
†: EA with soft genetic operators/Breeder GA [EASY/BGA] [50].
\( f^\star \): Function values at which the algorithms stop, but not the precise values of the global optimum.
#eval: Number of function evaluations.
n: Dimensionality of the problem.
techniques (BGA and EASY). However, the performance of the algorithm on the Shekel family is comparatively poorer than the existing techniques due to its golf-hole-like landscape. Function S5 has five prominent optima resting on a plateau, all having similar basin of attractions. Landscapes of this kind provide no useful information for guidance, rendering our proposed memory feature rather useless. Setting $\beta$ to high values casts no advantages to the search quality but in turn increases the computational resources needed. The drop in performance from S5 to S7 to S10 indicates that the rise in the number of prominent optima, which further supports the argument above, the more difficult this kind of functions is. For this kind of problems, we suggest a small $\beta$ value to decrease the remembrance in order to cut away the wastage of computational expenses. Also, strong competition effect among populations with strong cooperation within populations should be adopted. We discuss this in the next test.

4.3. Deception

In this test, we are going to demonstrate the capability of our algorithm in handling deceptive problems. Goldberg’s bipolar deceptive functions are designed for testing various GAs in this aspect [16]. The ability in solving deceptive problems is important to global search algorithms, as it indicates that an algorithm is not easily trapped in local optima. We tested our algorithm in handling these functions and showed that our algorithm is capable of solving deceptive problems (cf. [29] for details).

In addition to handling GA deceptive problems, our algorithm can also handle problems with golf-hole-like landscape such as those in the Shekel family—

<table>
<thead>
<tr>
<th>Problems</th>
<th>$f^a$ attained</th>
<th>#eval</th>
<th>%b</th>
<th>Conditions$^a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rb2</td>
<td>$-0.000000397$</td>
<td>2705.50</td>
<td>100</td>
<td>$\beta=0.90$, $N=50$</td>
</tr>
<tr>
<td>S5</td>
<td>10.004523</td>
<td>12,476.40</td>
<td>34</td>
<td>$\beta=0.97$, $N=50$</td>
</tr>
<tr>
<td>S7</td>
<td>10.100106</td>
<td>13,168.60</td>
<td>29</td>
<td>$\beta=0.97$, $N=50$</td>
</tr>
<tr>
<td>S10</td>
<td>10.126623</td>
<td>13,909.30</td>
<td>14</td>
<td>$\beta=0.97$, $N=50$</td>
</tr>
<tr>
<td>H3</td>
<td>3.861696</td>
<td>755.80</td>
<td>100</td>
<td>$\beta=0.90$, $N=30$</td>
</tr>
<tr>
<td>A30</td>
<td>$-0.00078$</td>
<td>18,679.68</td>
<td>100</td>
<td>$\beta=0.40$, $N=40$</td>
</tr>
<tr>
<td>A100</td>
<td>$-0.00074$</td>
<td>58,216.17</td>
<td>90</td>
<td>$\beta=0.35$, $N=40$</td>
</tr>
<tr>
<td>R20</td>
<td>$-0.48987$</td>
<td>5,413.22</td>
<td>100</td>
<td>$\beta=0.45$, $N=40$</td>
</tr>
<tr>
<td>R100</td>
<td>$-0.54718$</td>
<td>45,194.86</td>
<td>100</td>
<td>$\beta=0.45$, $N=40$</td>
</tr>
</tbody>
</table>

In this test result, the simplest version of our algorithm pBHS works well on most of the test problems in terms of the success rate, except Shekel’s functions (S5, S7 and S10).

$^a$ Percentage of runs reaching or exceeding the $f^+$ stated in Table 5.

$^b$ Hundred independent consecutive runs, $l = 16$. 

In Table 6, Testing result for pcBHS on Rb2, S5, S7, S10, H3, H6, A30, A100, R20 and R100
S5, S7 and S10. The landscape of these three functions share a commonality which makes global search very difficult: several sub-optima with similar basin of attractions sitting on a large flat plateau. The sizes of these attractors are so similar that no useful information can be obtained about the location of the global solution. Without this information, our remembrance scheme is rendered useless. To solve this problem, we lowered the $\beta$ value significantly to reduce computational expenses and to raise the speed of local search. The number of population $G$ has also been increased to counter-balance the suction-like local exploitation and to provide search redundancy.

Our previous experiment, as shown in Table 6 using $G = 1$ and $\beta = 0.97$, acts as a control for this experiment. The settings of this experiment are listed in Table 7. In this experiment, we dropped $\beta$ drastically to 0.05 and varied the number of populations from 1 to 14. The results are listed in Table 8, Figs. 10–12. It clearly shows that by increasing the number of populations and lowering of the $\beta$, the success rate on solving this kind of problems can be increased, while keeping the computational expenses more or less the same.

Increasing the number of populations blindly cannot guarantee better performance. As indicated in Figs. 10–12, the success rate levels with a constant

---

**Table 7**

Shekel family: experiment conditions

<table>
<thead>
<tr>
<th>Problems</th>
<th>$f^+$</th>
<th>$G$</th>
<th>Conditions$^a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S5</td>
<td>9.9</td>
<td>1,2,3,4,5,6,7,8</td>
<td>$N = 100$</td>
</tr>
<tr>
<td>S7</td>
<td>9.9</td>
<td>1,2,3,4,5,6,7,8</td>
<td>$N = 100$</td>
</tr>
<tr>
<td>S10</td>
<td>9.9</td>
<td>1,2,4,6,8,10,12,14</td>
<td>$N = 140$</td>
</tr>
</tbody>
</table>

$^a$ Hundred consecutive independent runs, $\mu = 1$, $\beta = 0.05$.

---

**Table 8**

Testing result for pccBHS on Shekel family

<table>
<thead>
<tr>
<th>$f^+$</th>
<th>#eval</th>
<th>$f^+$</th>
<th>#eval</th>
<th>$f^+$</th>
<th>#eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>S5</td>
<td></td>
<td>S7</td>
<td></td>
<td>S10</td>
<td></td>
</tr>
<tr>
<td>$G$</td>
<td>$%$</td>
<td></td>
<td>$G$</td>
<td>$%$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>10.030556</td>
<td>1860.6</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>55</td>
<td>10.023210</td>
<td>5323.7</td>
<td>2</td>
<td>47</td>
</tr>
<tr>
<td>3</td>
<td>68</td>
<td>10.012363</td>
<td>6331.0</td>
<td>3</td>
<td>66</td>
</tr>
<tr>
<td>4</td>
<td>86</td>
<td>10.021969</td>
<td>7487.6</td>
<td>4</td>
<td>74</td>
</tr>
<tr>
<td>5</td>
<td>87</td>
<td>10.011920</td>
<td>9519.9</td>
<td>5</td>
<td>89</td>
</tr>
<tr>
<td>6</td>
<td>92</td>
<td>10.013254</td>
<td>11,991.6</td>
<td>6</td>
<td>92</td>
</tr>
<tr>
<td>7</td>
<td>99</td>
<td>10.099583</td>
<td>11,196.6</td>
<td>7</td>
<td>95</td>
</tr>
<tr>
<td>8</td>
<td>97</td>
<td>10.013743</td>
<td>12,717.8</td>
<td>8</td>
<td>99</td>
</tr>
</tbody>
</table>

$f^+$: Average function value reached.  
#eval: Number of function evaluations.  
$: Success rate.
rise in computational expenses up to certain number of populations. However, this optimal number of populations is problem-dependent and can only be determined by examining the problem directly.

Fig. 10. Shekel family S5: (a) Graph showing the effect of different number of subgroups on the percentage of runs getting the global optimum. (b) Graph showing the computational expenses on using different number of subgroups.

Fig. 11. Shekel family S7: (a) Graph showing the effect of different number of subgroups on the percentage of runs getting the global optimum. (b) Graph showing the computational expenses on using different number of subgroups.

Fig. 12. Shekel family S10: (a) Graph showing the effect of different number of subgroups on the percentage of runs getting the global optimum. (b) Graph showing the computational expenses on using different number of subgroups.
5. Discussion

We have presented a global searching algorithm with satisfactory experimental results. According to No Free Lunch Theorem for Search [52], the algorithms do have strong features and at the same time weak areas.

5.1. Search with memory

Searching with memory is one of the significant features of pccBHS. By accumulating past searching experience, a global picture of the search space can be created. Although collecting information about the search space through the search is not a unique feature of pccBHS, existing algorithms, such as PBIL (Population-based Incremental Learning) and cGA (Compact Genetic Algorithm) are two examples. However, our algorithm differs from PBIL and cGA in the following ways:

1. Propagation of search information
   • Both PBIL and cGA generates the next generation population based on the current population only.
   • pccBHS is more reliable because it is moderated by the remembrance factor—the population is generated using the memory accumulated since the beginning of the searching.

2. Usage of search information
   Both cGA and pccBHS generated a population using probability vector. They differ on how the vector is generated.
   • In cGA, the vector represents the last population only.
   • The vector in pccBHS represents the knowledge on the search space accumulated throughout the search.

5.2. Learning rate

Concerning the learning rate, our remembrance value $\beta$ is more general and powerful than the Learning Rate (LR) of PBIL, because our remembrance value is ever changing in an adaptive way. Individual bit position has its own remembrance value. The values are changed according to the convergence status of the corresponding bit. In other words, the speed of the search is not handcrafted, but is being adjusted adaptively. On the contrary, the LR in PBIL is fixed throughout the whole search run. Premature convergence is less often in pccBHS.

5.3. Relevancy to schema theorem

Holland’s schema theorem which talks about the expectation of schema propagation generation from generation on applying various genetic operators
onto bit strings. The design of pccBHS is to partition the search space and to continuously do sampling to collect global information. Although the algorithm is illustrated by bit strings, it is not base on Holland's schema theorem. The purpose of partitioning the search space is to provide structural basis for smoothing search space and resolution control. There is no schema defined in our algorithm. Moreover, the algorithm has no genetic operator counterpart at all.

5.4. Arbitrary solution precision and no over-exploitation

Other features of the algorithms that worth mentioning are: (1) Bit strings (or any structure that is used to model the problem) is fixed in size, but it is not difficult to extend the algorithm implementation to variable-length string. In this way, the precision of the solution can be made to be as high as possible; (2) The accumulation of searching information is done by collaboration of the whole population, both fit and not fit. There would be no winner-take-all. It has the advantage of avoiding over-exploitation.

5.5. Weakness on some high-dimensional problems

pccBHS is designed to cater for vast different types of global optimization problems. It suffers from some deficiencies that are common to algorithms of similar categories. One of those is poor performance on solving high-dimensional problems with strong dependency among dimensions. To cater for this difficulty, we suggest the following search strategy:

(1) A very quick search to get to know more of the problem at hand, such as a low remembrance value favoring fast convergence.
(2) A few such quick trials to tell something about the problem. If the solutions obtained are similar, the problem is not a tricky one. If the solutions end up in two sets of dissimilar solutions clearly, we know that it probably has two optima, one of which might be the global optimum. If the solutions are so dissimilar, it is probable that the problem is of golf-hole like. Then we might need the island-model.

5.6. Choosing parameters

There are two parameters $\beta$ and $\tau$ used in pccBHS. The remembrance value $\beta$ controls how much past memory is retained. A large $\beta$ means retaining more past searching information in the memory and incorporating smaller amount of the latest searching information.

- With $\beta$ equals to 1, no new information will be stored. Since the memory at the beginning of the run is empty, the search is equivalent to random search.
• With $\beta$ equals to 0, no past information will be retained. The search ends up depending on the last population only.

In general, the larger the value of $\beta$, the slower the convergence of the run but the more reliable the collected information and the higher the success rate.

We do not normally use 0.0 and 1.0 values for $\beta$. We tend to use large $\beta$ values: $0.85 \leq \beta \leq 0.95$ depending on the difficulty of the problem. The more difficult the problem, the larger the $\beta$ value used. Another parameter $\tau$ used in the algorithm is the convergence threshold. It is used to control the convergence rate of the adaptive remembrance scheme. If we use a low threshold, premature convergence would easily occur, but if we use a higher threshold, the run will be slowed down. In this paper, we have not addressed the detail analysis of these two parameters. Under our current study, we only have empirical rules to choose these values.

5.7. Applicability and future enhancement

We have illustrated pccBHS using simple and well-known test problems commonly used in testing global optimization algorithms. Although we have not studied nor worked on in detail how pccBHS can be applied to other types of optimization problem with different kinds of constraints. However, we think the applicability of pccBHS is not limited to numerical optimization problems with bounded constraints. The modeling of pccBHS depends on search space partitioning and the organization of partitions into a tree-like structure. For any problem with solution space that can be modeled into a tree-like structure, pccBHS is applicable.

In general, pccBHS is not a specific heuristic, but a meta-heuristics. Different components of the algorithm and modeling are open for options. For example, the elite generation and selection method described in this paper can be changed to different heuristics according to the problem to be handled.

6. Conclusion

In this paper, a new iterative stochastic searching algorithm called probabilistic cooperative–competitive binary hierarchical search (pccBHS) is proposed to solve global optimization problems.

The core of the algorithm consists of two processes—cooperation and competition, which interact throughout the search. With cooperation, local optimal can be found effectively by agents interacting with each other to exploit their gathered information; while with competition, the strong exploitation effect can be leveled off to find the global optimum by keeping repulsive force among populations of agents to maintain search diversity. Furthermore, one
The advantage of this competition model over existing techniques is that it does not need a pre-defined radius which inherently limits the algorithm from finding optima in different resolutions simultaneously [9].

In order to further increase solution quality, we introduce three promising features which are commonly lacking in existing search algorithms: (1) resolution controls, (2) smoothing of search space, and (3) introducing memory during the stochastic search.

Comparing the performance of our algorithm with the existing ones, we have the advantage of being versatile and computationally more economical and at the same time maintaining, and sometimes exceeding, the solution quality.

Acknowledgements

This work is supported in part by Earmarked Grants (CUHK4212/01EE and CUHK 4351/02E) from the University Grants Councils of the Hong Kong Special Administrative Region.

Appendix A. Algorithms

Algorithm 1. The Information Processing Cycle

Procedure INFORMATIONPROCESSINGCYCLE
   global environment ← Empty
   While not converged
      Loop
         For each searching agent do
            search result ← Search(global environment)
         End For
      End While
   End Procedure

Algorithm 2. PCBHS—Cooperative Fitness Assignment and local elites Updating

This procedure assigns fitness (cooperative fitness) to all solution fragments. In each subpopulation, if the best fragment is better than the elite fragment $x_{m}^e$, it becomes the new elite fragment. It should be noted that the current elite $x^e$ is not changed in this procedure.
Procedure COOPEVALUATION
For each subpopulation $P_m$, $0 \leq m < n$
/* $x^e_m$: Elite fragment of the $m$th subpopulation */
\[ x^e_m \leftarrow x^e_m \]

For each solution fragment $x_{m,i}$ in $P_m$
\[ x \leftarrow \text{replace the } m\text{th element of } x^e \text{ by } x_{m,i} \]
\[ cf(x_{m,i}, x^e) \leftarrow F(x) \]
if \[ cf(x_{m,i}, x^e) > cf(x^e_m, x^e) \] then
\[ x^e_m \leftarrow x_{m,i} \]
End if
End for
End for
End Procedure

Algorithm 3. COMPETITION IN pccBHS

This procedure calculates the interaction fitness of a binary string. A binary string will have fitness deduction if the subgroup it belongs to overlap with other subgroups. The more it is overlapped with a subgroups or the more subgroups that it is overlapped with, the more the fitness is deducted.

Procedure COMPETITION
For each subgroup $g_I$, $0 \leq I < I_{\text{max}}$
For each subgroup $g_J$, $0 \leq J < J_{\text{max}}$
If $I$ not equal to $J$ then
\[ F_m(g_I, g_J) = \max\{g_{m}^{\text{max}}, g_{j}^{\text{max}}\} - \min\{g_{m}^{\text{min}}, g_{j}^{\text{min}}\} \]
\[ O_m(g_I, g_J) = \min\{g_{m}^{\text{max}}, g_{j}^{\text{max}}\} - \max\{g_{m}^{\text{min}}, g_{j}^{\text{min}}\} \]
\[ D_m(g_I, g_J) = \frac{O_m(g_I, g_J)}{F_m(g_I, g_J)} \]
For each $x_{m,j}$, $0 \leq j < N$
\[ P_m(g_I, x_{m,j}) = \frac{|x_{m,j} - x^e_m|}{F_m(g_I, g_J)} \]
\[ R_m(g_I, x_{m,j}) = D_m(g_I, g_J) \times (1 - P_m(g_I, x_{m,j})) \]
\[ cf(x_{m,j}, x^e) = cf(x_{m,j}, x^e) \times (1 - R_m(g_I, x_{m,j})) \]
End for
End if
End for
End for
End Procedure
Algorithm 4. pccBHS—A Probabilistic Cooperative–Competitive Hierarchical Search

This pseudo-code summarizes the whole pccBHS algorithm. At the initialization phase, the global environment is set to empty. In each iteration, a population is generated using the global environment, evaluated by the CoopEvaluation procedure, further evaluated by the Competition procedure and finally updating the global environment.

Procedure pccBHS
   Initialize global environment
   While not converged
      Generate population using the global environment
      cf ← CoopEvaluation
      I ← Competition
      Update global environment using I
   End while
End Procedure

Appendix B. Testing functions

Rosenbrock2 Rb2 = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 where n = 2, X = [-2, 2]^n,

Rastrigin \( R_n = nA + \sum_{i=1}^{n} x_i^2 - A \cos(2\pi x_i) \)

where n = 20, 100, A = 10.0, -5.12 \( \leq x_i \leq 5.12, \)

Ackley \( A_n = 20 \exp \left( -0.2 \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}} \right) \)

\[- \exp \left( \frac{\sum_{i=1}^{n} \cos(2 \pi x_i)}{n} \right) \] where \(-30.0 \leq x_i \leq 30.0, \)

Shekel \( S = -\sum_{i=1}^{10} \frac{1}{(x - A_i)(x - A_i)^{\tau} + c_i} \) where \( X = [0, 10]^4, \)
\[
A = \begin{bmatrix}
4 & 4 & 4 & 4 \\
1 & 1 & 1 & 1 \\
8 & 8 & 8 & 8 \\
6 & 6 & 6 & 6 \\
3 & 7 & 3 & 7 \\
2 & 9 & 2 & 9 \\
5 & 5 & 3 & 3 \\
8 & 1 & 8 & 1 \\
6 & 2 & 6 & 2 \\
7 & 3.6 & 7 & 3.6
\end{bmatrix},
\]
\[
c = \begin{bmatrix}
.1 \\
.2 \\
.2 \\
.4 \\
.4 \\
.6 \\
.3 \\
.7 \\
.5 \\
.5
\end{bmatrix}.
\]

Rastrigin R2 = \(-x_1^2 - x_2^2 + \cos(18x_1) + \cos(18x_2)\)

where \(-1.0 \leq x_i \leq 1.0, i = 1, 2,\)

Goldstein–Price GP2

\[
= -[1 + (x_1 + x_2 + 1)^2(10 - 14x_1 + 2x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \\
\cdot [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]
\]

where \(-2.5 \leq x_i \leq 2.0, i = 1, 2,\)

Hartman H3 = \(-\sum_{i=1}^{4} c_i \exp\left( -\sum_{j=1}^{n} a_{i,j}(x_j - p_{i,j})^2 \right)\)

where \(0.0 \leq x_j \leq 1.0, 1 \leq j \leq 3,\)

\[
a = \begin{bmatrix}
0.36890 & 0.11700 & 0.26730 \\
0.46990 & 0.43870 & 0.74700 \\
0.10910 & 0.87320 & 0.55470 \\
0.03815 & 0.57430 & 0.88280
\end{bmatrix},
\]
\[
p = \begin{bmatrix}
3 & 10 & 30 \\
0.1 & 10 & 35 \\
3 & 10 & 30
\end{bmatrix},
\]
\[
c = \begin{bmatrix}
1 \\
1.2 \\
3 \\
3.2
\end{bmatrix}.
\]

Shekel S1 = \(\sum_{i=1}^{m} \frac{1}{(k_i(x - a_i))^2 + c_i}\), where \(0.0 \leq x_j \leq 10.0, 1 \leq j \leq n\)
and

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>3.040</td>
<td>1.098</td>
<td>0.674</td>
<td>3.537</td>
<td>6.173</td>
<td>8.679</td>
<td>4.503</td>
<td>3.328</td>
</tr>
<tr>
<td>$k_i$</td>
<td>2.983</td>
<td>2.378</td>
<td>2.439</td>
<td>1.168</td>
<td>2.406</td>
<td>1.236</td>
<td>2.868</td>
<td>1.378</td>
</tr>
<tr>
<td>$c_i$</td>
<td>0.192</td>
<td>0.140</td>
<td>0.127</td>
<td>0.132</td>
<td>0.125</td>
<td>0.189</td>
<td>0.187</td>
<td>0.171</td>
</tr>
</tbody>
</table>

References


