Two coding based adaptive parallel co-genetic algorithm with double agents structure

Yongming Li *, Xiaoping Zeng, Liang Han, Pin Wang

College of Communication Engineering, Chongqing University, Chongqing 400030, China

A R T I C L E   I N F O
Article history:
Received 30 April 2008
Received in revised form 12 October 2008
Accepted 17 April 2009
Available online 28 November 2009

Keywords:
Genetic algorithm
Multi-population
Agent
Chain-like agent structure
Global numerical optimization
Feature selection
Adaptability
Size of sub-population

A B S T R A C T
This paper systematically proposed a multi-population agent co-genetic algorithm with double chain-like agent structure (MPATCGA) to solve the problem of the low optimization precision and long optimization time of simple genetic algorithm in terms of two coding strategy. This algorithm adopted multi-population parallel searching mode, close chain-like agent structure, cycle chain-like agent structure, dynamic neighborhood competition, and improved crossover strategy to realize parallel optimization, and has the characteristics of high optimization precision and short optimization time. Besides, the size of each sub-population is adaptive. The characteristic is very competitive when dealing with imbalanced workload. In order to verify the optimization precision of this algorithm with binary coding, some popular benchmark test functions were used for comparing this algorithm and a popular agent genetic algorithm (MAGA). The experimental results show that MPATCGA has higher optimization precision and shorter optimization time than MAGA. Besides, in order to show the optimization performance of MPATCGA with real coding, the authors used it for feature selection problems as optimization algorithm and compared it with some other well-known GAs. The experimental results show that MPATCGA has higher optimization precision (feature selection precision). In order to show the performance of the adaptability of size of sub-populations, MPATCGA with sub-populations with same size and MPATCGA with sub-populations with different size are compared. The experimental results show that when the workload on different sub-populations becomes not same, the adaptability will adaptively change the size of different sub-population to obtain optimization precision as high as possible.

© 2009 Elsevier Ltd. All rights reserved.

1. Introduction

Optimization algorithm is very important for machine learning, complex combination optimization, feature selection, operational research, and so on. The characteristic of the genetic algorithm (GA) as a good optimization method is its implicit parallelism which is a result of the evolution and the hereditary-like process. Besides, it is more likely to jump out of local optima to search for global optima. However, with the searched space and dimensions increasing, the complexity of searching increases greatly, it becomes easier to fall into local optima. In order to improve its performance, many researchers performed many works on it, and proposed many modified GAs (Goldberg, 1989; Holland, 1992; Michalewicz and Fogel, 2000; Kherallah et al., 2008).

Srinivas and Patnaik (1994) proposed one GA with adaptive parameters (AGA); the parameters are related with fitness value of the whole population, thereby keeping the diversity of the whole population. Dun-wei et al. (2002) proposed one genetic algorithm that can effectively avoid similar individuals’ crossover (SFGA). However, this algorithm only considers the adaptive crossover process. Zhong et al. (2004) introduced lattice-like agent structure into GA, and modified the selection operator, crossover operator, and mutation operator, thereby proposing one multi-agent GA (multi-agent genetic algorithm, MAGA). The relative experimental results show that the MAGA can obtain better optimization results than some other popular GAs for some complex benchmark test functions. The proposition of MAGA shows the potential of the combination of agent structure and GA. However, in MAGA, there are some things needing to be improved. Firstly, the neighborhood competition selection process is not dynamic enough. When the neighboring agents compete with the current agent to obtain new agent, maybe some of the neighboring agents have been updated. With MAGA, the competition between the current agent and its neighboring agent are done no matter whether the neighboring agents are updated or not. Secondly, the four-neighborhood individuals’ structure is likely to let some individuals with high fitness values occupy too many nodes (i.e. local top advantage) over early. If current agent has higher fitness value than its neighboring agents, when the current agent competes with its neighboring...
agents, the current agent can replace its neighbors and occupy the nodes of its neighbors. So, it is hard to keep whole population’s diversity, thereby leading to over early convergence. Based on this, we did some improvements, proposed dynamic neighborhood competition operator and chain-like agent structure (cycle chain agent structure), thereby designing one new agent GA-dynamic chain-like agent genetic algorithm (CAGA) (Zeng et al., 2008). According to some popular benchmark functions, the optimization result is better than MAGA to some extent. During the research of CAGA, we found that the improvement on time cost is still limited. For the complex optimization problems, even if we combine the characteristics of search space into optimization algorithm to reduce the algorithm’s time cost, the improvement on time cost is still limited and is not very apparent. In order to reduce the time cost and improve the optimization speed greatly, multi-population genetic algorithm is a good choice to be applied to realize parallel optimization. We can adopt multi-CUs (computing processing unit, CPU) to realize the multi-population genetic algorithm, each CPU realize one population. Apparently, the time cost will be reduced greatly. Currently, there are two realization modes for multi-population genetic algorithm. One kind of realization mode is to decompose the optimization problem into many sub-problems; every sub-problem uses one GA with single population. The shortcomings of the mode are in: firstly, it is hard to know whether the decomposition is reasonable. Secondly, because the optimization result of every genetic algorithm is optimal partial solution, the assembly of the optimal partial solutions into one optimal full solution is necessary. However, the assembly is very hard and skillful for the corresponding designer. Usually, the simple combination of all optimal partial solutions is not one optimal full solution for the whole optimization problem. Another kind of realization mode is to design one multi-population genetic algorithm, every sub-population searches optimal full solution in parallel, and the sub-populations exchange evolution information each other during genetic operation to search for global optimal full solution. Olsson (2001) proposed the co-evolutionary search algorithm in asymmetric space, obtaining satisfying results to some extent, but the searching speed and precision for complex searched space are not satisfying enough. Potter and De Jong (2000) proposed one collaboration co-genetic algorithm. They introduced multi-population idea into genetic algorithm to improve GA’s optimization performance. However, it just considers collaboration between sub-populations but does not consider competition between sub-populations. Ying et al. (2006) proposed one collaboration co-particle swarm optimization algorithm, but the algorithm has similar drawbacks as the one in the paper (Potter and De Jong, 2000). Su and Hou (2008) proposed another multi-population genetic algorithm for parameter optimization problem. The GA has two sub-populations, each sub-population optimizes different object function and two optimization results within one generation are obtained. After that, the two results are put together to make an optimal full solution. So it still belongs to the first realization mode discussed above.

Based on the analysis above, this paper proposes one multi-population co-genetic algorithm with double chain-like agent structure (close chain-like agent structure and cycle chain-like agent structure), combining chain-like agent structure and multi-population parallel searching. In this algorithm, inside every sub-population, close chain-like agent structure is applied; the individuals of some sub-population are connected with one close chain as agents. Every sub-population is connected with one cycle chain and shares some common agents (they are called shared agents). The sub-populations evolve themselves and cooperate and compete with each other. Besides, the genetic operators are improved, they include dynamic neighborhood competition operator, neighborhood orthogonal crossover operator and adaptive mutation operator to effectively keep and enhance the diversity of the sub-population, being good to searching for global optima in complex and high dimensional search space.

Except global numerical optimization problems, GAs are often used for feature selection problems as optimization algorithm. Feature selection problems are themselves optimization problems; they can be looked as searching of optimal subset (optimal solution). Therefore, as good optimization algorithm, GA is widely used for feature selection problems. The feature selection performance (feature selection precision and speed) can evaluate the performance of some GA or some other optimization algorithm. Therefore, in this paper, we will discuss how to use MPATCGA for feature selection and organize some feature selection experiments to evaluate the performance of MPATCGA.

In our previous work, based on global optimization problems, we have proposed a similar modified genetic algorithm (MPAGA) with real coding (Li and Zeng). Based on feature selection problem, we have proposed a similar modified genetic algorithm (MPAGAFS) with binary coding (Yongming et al.). But, our motivation to write this paper is in that, firstly, the two modified genetic algorithm has same agent structure, so it is necessary to discuss the two genetic algorithms together in order to let the interested authors know what is same thing and difference between the two genetic algorithms and how to use them correctly. Secondly, the genetic algorithms are parallel agent genetic algorithms; all the sub-populations can search in parallel. In the real word, the task (search space) on each sub-population cannot be same. Therefore, the size of sub-population should be adaptive. This paper modifies our previous work with the self-adaptability and gives the relevant experimental results. Thirdly, we slightly modify the dynamic competition strategy with very similar time cost and slight improved precision. Fourthly, for global optimization, some experiments need to be further done; therefore, we give the more experimental results in this paper. Fifthly, for feature selection, some experiments need to be further done, such as multi-class dataset; therefore, we give the more experimental results in this paper.

The paper is organized as follows: Section 2 describes the process of the design of the MPATCGA. In Section 3, we prove that the MPATCGA can be better than other optimization algorithm for some optimization problems in terms of NFL theorem. In Section 4, we evaluate the efficiency of the proposed MPATCGA for numerical optimization and feature selection through comparing it with some other popular genetic algorithms. Finally, some conclusions are offered in Section 4.

2. Two coding based multi-population co-genetic algorithm with double chain-like agents structure (MPATCGA)

2.1. Multi-population cycle chain-like agent structure

The model of optimization problem of functions can be expressed as follows:

$$\min_{\vec{x}} f(\vec{x}) \text{ satisfy } \vec{B}_l \leq \vec{B}_u$$  \hspace{1cm} (1)

where, $\vec{x} = (x_1, x_2, \ldots, x_n)$ is some variable vector in variable domain $\mathbb{R}^n$, $f(\vec{x})$ is object function, $\vec{B}_l = (B_{l1}, B_{l2}, \ldots, B_{ln})$, $\vec{B}_u = (B_{u1}, B_{u2}, \ldots, B_{un})$ are the upper and lower boundary (that is the definition domain of $x_i$ $[B_{li}, B_{ui}]$).

Multi-population cycle chain-like agent structure means that in terms of the position information of agents, the whole population is divided into some sub-populations. The agents...
inside sub-population are located in close chain-like agent structure and cooperate with each other. Each sub-population is connected with other sub-population through ‘shared agents’ and in the form of cycle chain-like agent structure; they cooperate with each other though sharing the information of ‘shared agents’. Suppose the number of shared agents is $S$, the agent that is located in the $j$th node in the $i$th sub-population is expressed as $L_{ij}$, where $i = 1, 2, \ldots, M$, $j = 1, 2, \ldots, L$. The neighborhood domain of $L_{ij}$ is defined as $\text{Neibor}_{i_j} = \{L_{ij}, L_{i_{j+1}}, L_{i_{j-1}}\}$, where

$$L_{ij} = \begin{cases} L_{i1} & j = 1 \\ L_{i_{j-1}} & j \neq 1 \\ L_{i_{j+1}} & j \neq L \\ \end{cases}$$

(2)

Fig. 1 shows the multi-population cycle chain-like agent structure with 6 agents per sub-population and 2 shared agents. The motivation of the agents for evolution is to augment their power, so they cooperate and compete with each other. Finally, the agent with low power will die, and new agent will occupy its position. Inside the sub-population, the cooperation and competition take place between the agent and its neighbors, the introduction of the shared agents will supply the genetic information of other sub-populations, thereby improving the efficiency of the evolution. Within the structure, each ring represents a ring-like agent structure. In the ring-like agent structure (close chain-like agent structure), all the agents live in a close chain-like environment, $L$, which is called an agent ring. The size of $L$ is $1 \times L_{\text{size}}$, where $L_{\text{size}}$ is an integer, 1 means one dimensional agent structure. Each agent is fixed on a ring-point and it can only interact with its neighbors (Fig. 2).

2.2. Genetic operators

2.2.1. Dynamic neighborhood competition selection operator

**Definition of the energy**: an agent, $a$, represents a candidate solution to the optimization problem in process. The value of its energy is defined as follows:

$$\text{Eng}(L_{1i}) = \text{fitness}(L_{1i})$$

(3)

where $\text{fitness}()$ means the fitness value of some individual in the population. For numerical optimization, $\text{fitness}()$ means the

![Fig. 1. Multi-population cycle chain-like agent structure.](image1)

![Fig. 2. Tested functions: (a) $f_3$ and (b) $f_4$.](image2)
corresponding function needing to be optimization; for feature selection, it corresponds to some evaluation criteria.

As can be seen, each agent stands for an individual. In order to realize the local perceptivity of agents, the environment is constructed as a chain-like structure as mentioned above.

Suppose that the current agent is located at \(L_1, L_1=(l_{1,1}, l_{1,2}, \ldots, l_{1,n})\). \(\text{Max}_i = (m_{i,1}, m_{i,2}, \ldots, m_{i,n})\) is the agent with maximum energy among the neighbors of \(L_1\), where \(n\) means the number of genes. \(l_{i,n}\) means the \(n\)th gene of \(i\)th individual \(L_1\) (that is chromosome), \(m_{i,n}\) means the \(n\)th gene of \(\text{Max}_i\). That is, \(\text{Max}_i \in \text{Neibors}, j\) and \(v_a \in \text{Neibors}_i\), then \(\text{Eng}(a) \leq \text{Eng}(\text{Max}_i)\).

If \(L_1\) satisfies formula (5), then it still live in the agent chain. Or else, it will die, and its chain-point will be occupied by \(\text{New}_i\):

\[
\text{Eng}(L_{1,j}) \geq \text{Eng}(\text{Max}_{1,j}) \tag{4}
\]

**Dynamic competition strategy**: During competition process, the \(\text{Max}_{1,j} = \max (L_{1,j}, L_{1,j})\). The competition process is done in ascending order, after the competition of the 1st agent, the 1st agent is updated. Assuming the \(i\)th agents before competition and after competition are \(L_{i,j}^{\text{pre}}\) and \(L_{i,j}\), respectively, so \(\text{Max}_{1,j}\) is determined by the following procedures:

\[
\begin{align*}
\text{if } i &= 1 \\
\text{then } \text{Max}_{1,j} &= \max(L_{1,j}^{\text{pre}}, L_{1,j}^{\text{pre}} - L_{1,j}^{\text{pre}} + L_{1,j}^{\text{pre}}) \\
\text{else } i &= \text{size} \\
\text{then } \text{Max}_{1,j} &= \max(L_{i,j}^{\text{post}}, L_{i,j}^{\text{post}} - L_{i,j}^{\text{post}} + L_{i,j}^{\text{post}}) \\
\text{else } \text{Max}_{1,j} &= \max(L_{i,j}^{\text{post}}, L_{i,j}^{\text{post}} - L_{i,j}^{\text{post}} + L_{i,j}^{\text{post}})
\end{align*}
\]

Within one sub-population, when the \(i\)th agent is called, if any of the neighboring agents of the \(i\)th agent is updated, the after updated agent rather than before updated agents cooperate with the current agent to obtain the new agent \(\text{Max}_{1,j}\). For example, if 5th agent is called. The neighboring agents of the current agent are 4th agent and 6th agent. The before updating 4th agent is \(L_{1,j}^{\text{pre}}\), the before 6th agent is \(L_{1,j}^{\text{pre}}\), the after updated 4th agent is \(L_{1,j}^{\text{post}}\) and the after updated 6th agent is \(L_{1,j}^{\text{post}}\), if the 4th agent is updated when the 5th agent is being processed, traditional method is that the \(L_{1,j}^{\text{pre}}, L_{1,j}^{\text{pre}}, L_{1,j}^{\text{post}}, L_{1,j}^{\text{post}}\) compete each other to obtain \(\text{Max}_{1,j}\). However, with the dynamic competition strategy, the corresponding method is that the \(L_{1,j}^{\text{post}}, L_{1,j}^{\text{pre}}\) and \(L_{1,j}^{\text{post}}, L_{1,j}^{\text{post}}\) compete each other to obtain \(\text{Max}_{1,j}\). The major advantage is that with similar computational complexity, the 4th agent is used twice and dynamically, the efficiency is higher.

### 2.2.1.1. Discussion on selection operator with real coding.

Normally, the operator with binary coding is for feature selection and is described below.

The neighborhood competition selection operator is described as follows: Suppose the order of competition selection is from left to right, the current agent is \(L_{1,j}\), the neighbors are \(\text{Nbs}_{1,j}\), \(\text{Nbs}_{1,j} = \{L_{1,j}, L_{1,j}, \ldots\}\), \(i=1,2,\ldots \text{popsize}\). Updating of \(L_{1,j}\) is as the following formula:

\[
\begin{align*}
L_{1,j}^{\text{new}} &= \begin{cases} \\
L_{1,j}^{\text{new}} & \text{fitness}(L_{1,j}^{\text{new}}) > \text{fitness}(\text{max}(L_{1,j}, L_{1,j})) \\
L_{1,j}^{\text{new}} & \text{fitness}(L_{1,j}^{\text{new}}) = \text{fitness}(\text{max}(L_{1,j}, L_{1,j})) \\
L_{1,j}^{\text{new}} & \text{fitness}(L_{1,j}^{\text{new}}) < \text{fitness}(\text{max}(L_{1,j}, L_{1,j}))
\end{cases}
\end{align*}
\]

In formula (9), \(c_{i,j}\) means competition selection between agent \(L_{1,j}\) and \(L_{1,j}\), the two agents consist of lots of genes:

\[
\begin{align*}
L_{1,j} &= (c_{i,j}, c_{i,j}, \ldots, c_{i,j}, \ldots, c_{i,j}) \\
L_{1,j} &= (c_{i,j}, c_{i,j}, \ldots, c_{i,j}, \ldots, c_{i,j})
\end{align*}
\]

\(c_{i,j}\) means \(j\)th gene of \(L_{1,j}\), \(c_{i,j}\) means \(j\)th gene of \(L_{1,j}\), length means number of genes of single agent. The competition selection between agent \(L_{1,j}\) and \(L_{1,j}\) can be called \(L_{1,j}, L_{1,j}\), the processing is as follows:

If \(c_{i,j} = c_{i,j}\), then \(c_{i,j}\) no change; elseif \(c_{i,j} = c_{i,j}\), then \(c_{i,j} = U(0, 1)\).

If real coding, \(c_{i,j} = U(0, 1)\), else binary coding, \(c_{i,j}\) = round(U(0, 1)).

Where, \(U(0, 1)\) means random number generator and is within the domain [0, 1].

### 2.2.2. Neighborhood orthogonal crossover operator

Two crossover operators for binary coding and real coding are described here, respectively. Orthogonal crossover operator with real coding can be used for numerical optimization, and adaptive crossover operator with binary coding can be used for feature selection. Both the two operators are used between some agent and its neighboring agents.
2.2.2.1. Neighborhood orthogonal crossover operator with real coding

The orthogonal crossover operator was described in Leung and Wang (2001). An orthogonal array can specify a small number of individuals that are scattered uniformly in the search space; the orthogonal crossover operator can generate a small but representative sample of potential individuals. In CAGA, this operator is performed between \( L_{1j} \) and \( \text{Max}_{1j} \) to achieve the both sides’ cooperation. Combining this paper, the design of this operator is described briefly as follows, and for details, please see Leung and Wang (2001).

Assuming that the search space defined by \( L_{1j} \) and \( \text{Max}_{1j} \) is \([x_{1j}, x_{nj}]\) as follows:

\[
\begin{align*}
    x_{ij} &= (\min(l_{1j}, m_{1j}), \min(l_{2j}, m_{2j}), \ldots, \min(l_{nj}, m_{nj})) \\
    x_{iu} &= (\max(l_{1j}, m_{1j}), \max(l_{2j}, m_{2j}), \ldots, \max(l_{nj}, m_{nj}))
\end{align*}
\]

The domain of the \( b \)th dimension is quantized into \( \beta_{b1}, \beta_{b2}, \ldots, \beta_{bQ2} \) as follows:

\[
\begin{align*}
    \beta_{b1} &= \left\{ \begin{array}{ll}
    \min(l_{b}, m_{b}) & j = 1 \\
    \min(l_{b}, m_{b}) + (j-1) \left( \frac{l_{b} - m_{b}}{Q_{2} - 1} \right) & 2 \leq j \leq Q_{2} - 1 \\
    \max(l_{b}, m_{b}) & j = Q_{2}
    \end{array} \right.
\end{align*}
\]

Specially, we randomly generate \( F - 1 \) integers \( k_{1}, k_{2}, \ldots, k_{F-1} \) such that \( 1 < k_{1} < k_{2} \ldots < k_{F-1} < N \), and then create the following \( F \) factors for any chromosome \( a = (x_{1j}, x_{nj}) \):

\[
\begin{align*}
    f_{1} &= (x_{1j}, \ldots, x_{nj}) \\
    f_{2} &= (x_{k_{1}j+1}, \ldots, x_{nj}) \\
    &\vdots \\
    f_{F} &= (x_{k_{F-1}j}, \ldots, x_{nj})
\end{align*}
\]

The \( f_{b} \) th factor \( f_{b} \) with \( Q_{2} \) factor is determined as follows:

\[
\begin{align*}
    f_{b}(1) &= (\beta_{b,k_{1}-1+1}, \beta_{b,k_{1}-1+2}, \ldots, \beta_{b,k_{1}}) \\
    f_{b}(2) &= (\beta_{b,k_{1}+1}, \beta_{b,k_{1}+2}, \ldots, \beta_{b,k_{2}}) \\
    &\vdots \\
    f_{b}(Q_{2}) &= (\beta_{b,k_{F-1}+1}, \beta_{b,k_{F-1}+2}, \ldots, \beta_{b,Q_{2}})
\end{align*}
\]

The orthogonal matrix \( M_{d}(Q_{2}) = [b_{ij}]_{d_{i}f} \) is used to generate the following \( M_{2} \) chromosomes (here they are called as agents):

\[
\begin{align*}
    &\{ (f_{1}(i), f_{2}(i), \ldots, f_{F}(i)) \} \\
    &\{ (f_{1}(i), f_{2}(i), \ldots, f_{F}(i)) \} \\
    &\vdots
\end{align*}
\]

Finally, among the \( M_{2} \) agents, the one with the biggest energy is selected to replace \( L_{1j} \).

2.2.2.2. Adaptive neighborhood crossover operator with binary coding

In the crossover process, the crossover probability \( p_{c,i} \) is calculated adaptively. The corresponding formula is as follows:

\[
P_{c,i} = \left\{ \begin{array}{ll}
    \left( \frac{f_{\text{max}} - f_{\text{ave}}}{f_{\text{max}} - f_{\text{ave}}} \right)^{1/\text{GH}(i,t)} & f_{\text{ave}} \geq f_{\text{ave}} \\
    1 & f_{\text{ave}} < f_{\text{ave}}
    \end{array} \right.
\]

Here, \( p_{c,i} \) means the probability of crossover about the crossover operation between the \( L_{1j} \) and \( \text{Max}_{1j} \). \( \text{GH}(i,t) \) means the distance between the \( L_{1j} \) and \( \text{Max}_{1j} \). \( f_{\text{max}} \) means the maximum value of both the individuals, \( f_{\text{ave}} \) means the average fitness value of all the individuals in the current population, \( f_{\text{ave}} \) means the average fitness value of all the individuals. The crossover procedure is as follows:

if \( U(0,1) < p_{c,i} \) do single point crossover processing between \( L_{1j} \) and \( \text{Max}_{1j} \) else keep \( L_{1j} \) no change

2.2.3. Adaptive mutation operator

Two mutation operators for binary coding and real coding are described here, respectively. Adaptive mutation operator with binary coding can be used for feature selection, and adaptive crossover operator with real coding can be used for numerical optimization.

2.2.3.1. Adaptive mutation operator with real coding. A new agent, \( \text{New}_{1} = (n_{e_{1}}, n_{e_{2}}, \ldots, n_{e_{n}}) \) is generated as

\[
\text{New}_{1} = \begin{cases}
    l_{ik} + \frac{1}{t} \cdot \text{U}(0,1) & U(0,1) < p_{m} \\
    l_{ik} & \text{else}
\end{cases}
\]

where \( p_{m} \) means mutation probability, \( \text{G}(0,(1/t)) \) is a Gaussian random number generator, \( t \) is the evolution generation. According to Michalewicz and Fogel (2000), the suitable \( p_{m} \) is advised to be set as \( 1/n \).

2.2.3.2. Adaptive mutation operator with binary coding. In the crossover process, the mutation probability \( p_{m} \) is calculated adaptively based on the length of chromosome (Zhong et al., 2004). The crossover procedure is as follows:

if \( U(0,1) < 1/n \) do single point mutation processing between \( L_{1j} \) (namely, some gene changes its value from 1 to 0 or vice versa randomly) else keep \( L_{1j} \) no change

where \( n \) means number of genes

2.3. Stopping criterion

\( f_{\text{ave}} \) can reflect the evolution of the current population. \( f_{\text{best}} \) stands for the best average fitness value since beginning. \( k_{\text{stop}} \) means a counter, it counts the number that \( f_{\text{best}} \) has no change. If \( k_{\text{stop}} > k \), the search stops. The setting of \( k \) is described in experiments section.

2.4. Elitism strategy

Agents have knowledge which is related with the problem that they are designed to solve. With elitism strategy, the agent can inherit the good solution from the former generation. This method can make the best solution within ith generation better than or equals to the best solution in the former \((i-1)\) generations. In order to avoid repetition, the detailed operation is omitted here; it can be found in Section 3.

2.5. Self-adaptability of MPATCGA

As discussed above, the MPATCGA can realize the parallel optimization because of its multiple sub-populations. In the real problem, it cannot be guaranteed that the workload on each sub-population is same. For some sub-population, when its workload become smaller, if the size of the sub-population remains same, the precision does not become better, but the time cost will become more. When its workload become heavier, if the size of the sub-population remains same, the time cost does not become less, but the precision will become worse since it seems the number of agents is not enough.

Based on the discussion, the self-adaptability of MPATCGA is applied into the MPATCGA. As we know, an evaluation criterion is needed to guide automatic modification of size of sub-population. The evaluation criterion is diversity measurement of sub-population and described as follows:

\[
\text{div}_{sub} = \frac{\sum_{i=1}^{n} |f_{i,k} - F_{i,k}|}{n}
\]
where the $f_{ij}$ means value of the $j$th agent in the $i$th sub-population; $f_i$ means the average value of all the agents within the $i$th sub-population; $n_i$ means the size of the $i$th sub-population (number of the agents in the $i$th sub-population). When the $div_{sub}$ becomes lower than some threshold $T$, the size of sub-population becomes smaller with an agent (i.e. number of the agents minus 1); when the $div_{sub}$ becomes higher than some threshold $T$, the size of sub-population becomes larger (i.e. number of the agents plus 1). Our rationale is that if workload on some sub-population becomes heavier, the diversity of the agents in the sub-population becomes larger. Therefore the size of the sub-population needs to be increased. If workload on some sub-population becomes smaller, it means the search space for the sub-population becomes smaller; the diversity of the agents in the sub-population becomes smaller. Therefore the size of the sub-population needs to be decreased.

2.6. Realization of MPATCGA algorithm

The MPATCGA algorithm can be described as follows.

Procedure of MPATCGA

Notes: Suppose the best individual in the whole population in the $i$th generation is $i_{best\_whole}$, the number of evolution counter is $k_{cut\_whole}$, the upper boundary of $k_{cut\_whole}$ is TIMES\_OUT. The stopping criterion here is as follows: compare the $i_{best\_whole}$ and $i_{cut\_whole}$; if the difference of them is lower than $\delta$, the $k_{cut\_whole}$ is added with 1, or else, the $k_{cut\_whole}$ is updated with 0. When the $k_{cut\_whole}$ equals to TIMES\_OUT, quit the whole evolution and output the final optimization.

Begin

Initialization: Randomly generate an initial population: (real coding for numerical optimization, binary coding for feature selection)

Sub-population division: divide initial population into $M$ sub-populations based on the size of $L$;

While (stopping criteria are not satisfied)

Each sub-population evolves, respectively, based on principle of MPATCGA\_IN;

Judge whether all the sub-populations finish their one generation evolution, if so, the $M$ best individuals are obtained;

Judge the $M$ best individuals and obtain the best individual in the whole population in the current generation;

Implement Elitism strategy;

End While

End

Procedure of MPATCGA\_IN

Notes: In MPATCGA\_IN, the neighborhood competition operator is applied on each agent. As a result, the agents with lower energy are cleaned out from the agent chain so that there is more developing space for the promising agents. The neighboring crossover operator and the mutation operator are applied on each agent, respectively. At the end of $i$th generation, the best agent in this generation competes with the best agent in $(i-1)\_th$ generation, and $i_{best\_sub}$ (the best agent during $t$ generations' evolution) is updated.

Begin

initialize $L^0$, update pop$^0_{best\_sub}$, and $t--0$;

While (stopping criteria are not satisfied)

If self-adaptability enabled, then automatic change size of sub-population; else do not change size of sub-population do dynamic neighboring competitive selection processing and update $L^t$, obtaining $L^{t+1/3}$;

for each agent in $L^{t+1/3}$, do crossover processing on it, obtaining $L^{t+2/3}$;

for each agent in $L^{t+2/3}$, do mutation processing on it, obtaining $L^{t+1}$;

find inde$^t_{best\_sub}$, in $L^{t+1}$, and compare inde$^t_{best\_sub}$ and inde$^{t-1}_{best\_sub}$, if $Eng(ind_{best\_sub}^{t+1/3}) > Eng(ind_{best\_sub}^{t+2/3})$, then

$ind_{best\_sub} = ind_{best\_sub}^{t+1/3}$, else

$ind_{best\_sub}^{t+1/3} = ind_{best\_sub}^{t+1/3} + 1/3$, do crossover processing on it, obtaining $L^{t+1/3}$;

$ind_{best\_sub}^{t+1/3} = ind_{best\_sub}^{t+1/3} + 2/3$, do mutation processing on it, obtaining $L^{t+2/3}$;

End While

End

Comment: $L^t$ represents the agent chain in the $t$th generation, and $L^{t+1/3}$ and $L^{t+2/3}$ are the mid-chains between $L^t$ and $L^{t+1}$. $L^{t+1}$ is the agent chain after mutation processing in the $t$th generation. $i_{best\_sub}$ is the best agent among $\{L^0, L^1, ..., L^t\}$, and $i_{best\_sub}$ is the best agent in $L^t$. $p_c$ and $p_m$ are the probabilities to perform the neighboring crossover processing and the mutation processing.

The optimization precision and time cost are two important indices to show the performances of optimization algorithms. Whether MPATCGA can have better optimization precision is still unknown. We did many modifications to enhance its optimization performance: Dynamic neighborhood competition operator is similar to as the principle of "good ones win and bad ones lose" in the nature. The individuals with high fitness values are kept, the individuals with low fitness values are not kicked out simply, but are improved with their neighbors to obtain new individuals, and the diversity of the whole population is kept and enhanced. Neighborhood orthogonal operator can obtain different individuals as possible, thereby keeping and enhancing the diversity of the population. Besides, with the shared agents, the sub-populations can share genetic information with each other, the optimization precision can be assured. Besides, the optimization result obtained by any sub-population is a full solution, so it is not necessary to consider how to combine the partial solutions into a full solution, and the error occurred during combination of a full solution can be avoided. However, the modifications cannot prove its advantage of optimization precision directly. Therefore, it is very necessary to verify the optimization precision of the algorithm and compare it with other popular GA through empirical insights.

2.7. Computational complexity

As we know, the MPATCGA can realize parallel optimization. Suppose each CPU (computing processing unit) implements one sub-population, CPU shares genetic information with other CPUs through shared agents. Compared with the time cost needed for genetic operation, the time cost for exchanging genetic information is little and can be neglected. Suppose the time cost needed by each sub-population is equal, the time cost with each CPU should be average time cost $TimeAvg: TimeAvg = Time/kn_{sub}$, where $Time$ means the time cost of whole population with MPATCGA, $kn_{sub}$ means number of sub-populations. Suppose the time complexity of MAGA is $O(gp)$, if MPATCGA is realized by multiple CPUs in parallel, the time complexity of MPATCGA is $O(gp/kn_{sub})$ approximately, where $g$ means the generations of iteration, $p$ means the size of whole population.

2.8. MPATCGA and the NFL theorem

Briefly speaking, the no-free-lunch (NFL) theorem for optimization proposed by Wolpert and Macready states that any two blackbox algorithms (deterministic or stochastic) have the same
average performance over all problems (Wolpert and Macready, 1997). Random search is one such blackbox algorithm, so all blackbox algorithms have the same average performance as random search over all problems. One immediate implication of this theorem means that when a new algorithm is shown to perform better than others over some problems, then the new algorithm will perform worse over all remaining problems on average.

However, it has been seen that in practice one does not need an algorithm that performs well on all possible functions, but only on a subset that arises from the constraints of real-world problems. For example, it has been shown that for pseudo-Boolean functions restrictions of the complexity can lead to subsets of functions on which some algorithms perform better than others (Whitley, 1999). Therefore, if we can prove the NFL theorem cannot be applied in the functions (or problems) under investigation with the way.

As we know, Whitley has proven that NFL results hold for any subset $F$ of the set of all possible functions if and only if $F$ is closed under permutation (c.u.p.) (Schumacher et al., 2001). Based on this important result, classes of functions where NFL does not hold can be derived simply by proving that these classes are not c.u.p.

The functions under investigation by this paper are from the popular benchmark functions (such as those in Yao et al., 1999). These functions are multimodal functions, and can verify the searching performance of the given optimization algorithms, therefore they are used in many researches widely (for example, seen in Zhong et al., 2004; Pan and Kang, 1997; Tu and Lu, 2004, and so on). Fig. 3 shows the functions under 2 dimensions (will be used in Section 3). As the figure shows, in the two functions, there are many local minima, but the practical number of local minima of any one of these functions is smaller than the maximal possible number. Without question, if we prove that these kinds of functions are not c.u.p, then the NFL theorem is invalid here.

Let us do some necessary definitions. We consider a finite search space $X$ and a finite set of totally ordered cost values $Y$. Let $\mathcal{F} = Y^X$ be the set of all objective functions $f : X \to Y$ to be optimized (also called fitness, energy, or cost functions). Given a function $f$ and a neighborhood relation on $X$, we define $\max_{f\in\mathcal{F}}(f)$ as the maximal number of minima that functions with the same $Y$-histogram as $f$ can have (i.e. functions where the number of $X$-values that are mapped to a certain $Y$-value are the same as for $f$). Thus, it follows:

**Theorem.** If the number of local minima of every function $f$ in a nonempty subset $F \subset \mathcal{F}$ is constrained to be smaller than the maximal possible $\max_{f\in\mathcal{F}}(f)$, then $F$ is not c.u.p.

**Proof.** Let $g = \arg \max_{f \in \mathcal{F}}(f)$ be a function with maximal local minima, $s_{\max}(f)$ means the practical maximal local minima. Apparently, for these functions here, $s_{\max}(f) < \max_{f \in \mathcal{F}}(f)$.

If $F$ is c.u.p (that means any two functions $f$, $g$ have the same $Y$-histogram).

Then the function $g\pi$ is in $F$, there exists permutation $\pi \in \Pi(X)$ (that means $f\pi = g$).

This function has $s_{\max}(f) = s_{\max}(g\pi) = \max_{f \in \mathcal{F}}(g) = \max_{f \in \mathcal{F}}(f)$, which contradicts the local minima constraint.

---

**Fig. 3.** The comparison of convergence performance of MPATCGA and MAGA under different dimensions: (a) 2 dimensions, (b) 10 dimensions, (c) 50 dimensions and (d) 100 dimensions.
Table 1
Test functions ($f_{\text{min}}$ is the relevant global optima, $n$ is dimensions of test functions, SD is searched domain).

<table>
<thead>
<tr>
<th>Test functions</th>
<th>SD</th>
<th>$f_{\text{min}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1(x) = \sum_{i=1}^{n} x_i \sin(\sqrt{n}x_i)$</td>
<td>$[-500,500]$</td>
<td>$-418.95n$</td>
</tr>
<tr>
<td>$f_2(x) = \sum_{i=1}^{n} x_i^2 - 10 \cos(2\pi x_i) + 0.1$</td>
<td>$[-12.5,12]$</td>
<td>0</td>
</tr>
<tr>
<td>$f_3(x) = -20 \exp\left(-0.2 \sum_{i=1}^{n} x_i^2\right) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$</td>
<td>$[-32,32]$</td>
<td>0</td>
</tr>
<tr>
<td>$f_4(x) = \frac{1}{n} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{n}}\right) + 1$</td>
<td>$[-600,600]$</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2
Number of sub-populations with different shared agents and size of sub-population.

<table>
<thead>
<tr>
<th>Size of sub-population</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of shared agents</td>
<td>1</td>
<td>21</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>63</td>
<td>21</td>
</tr>
</tbody>
</table>

Therefore, $F$ is not c.u.p, the NFL theorem cannot be applied these functions. □

Some researchers have done several similar researches about the conditions that obviate the no free lunch theorems for optimization, interested authors can find more detailed content in their papers (for example, Koehler, 2007; Igel and Toussaint, 2003, and so on).

When the constraints are fixed and NFL does not apply, a given algorithm may be better or worse on average than random search. MPATCGA is such an algorithm. Whether it performs better or worse than random search for these functions is unknown on average. So we have to do experiments to verify it.

3. Experiments and analysis of results

From the flow chart of MPATCGA, it is seen that the algorithm can realize multi-sub-population parallel optimization, so the time cost can be reduced a lot. Suppose the number of shared agents is 1, the size of sub-population is $n$, and the size of whole population is $N$, the population can be divided into $N/(n-1)$ sub-populations with size of $n$. Compared with the time cost needed by genetic operation within each sub-population, the time cost of transmission of the values of the shared agents between neighboring sub-populations is a little. Therefore, theoretically the time cost of MPATCGA can be reduced to $(n-1)/N$ times of that of agent GA with single population. Without question, the parallel optimization can obtain faster optimization speed and lower time cost.

However, the optimization precision is very important and cannot be verified theoretically, so it needs to be verified through experiments. In order to show the optimization precision performance of this algorithm, global numerical optimization experiments and feature selection experiments were organized.

3.1. Global numerical optimization experiments

Some popular test functions were used in Table 1 for comparing MPATCGA and MAGA (Yao et al., 1999). $f_1$–$f_4$ are multimodal functions with many local optima (traps). Fig. 3 shows the test functions listed in Table 1. As the figure shows, the functions have a lot of local optima, which trap optimization method. It is not easy to find out the global optima or near-global optima. It means that the closer to the global optima, the better the algorithm will be.

The reasons for using MAGA for comparison are: firstly, it is also one agent GA with agent structure, the comparison of MPATCGA and MAGA can show the advantage of the agent structure of MPATCGA directly. Secondly, MAGA is an agent GA with single population, the comparison of MPATCGA and MAGA can show the MAPGA cannot only realize parallel optimization, thereby reducing the time cost needed greatly, but also obtain the precise optimization results as well as or even sometimes better than MAGA. Thirdly, the paper (Zhong et al., 2004) said that MAGA performed better than some well-known algorithms such as OGA/ Q, AEA, FEP, BGA, so the comparison with it can show MPATCGA better performance over those genetic algorithms indirectly.

Since the algorithm is parallel processing through multi-sub-population with smaller size, the corresponding runtime is smaller than the other genetic algorithm with single population including MAGA. However, the optimization precision of this algorithm needs to be evaluated. Therefore, lots of experiments about the optimization precision were organized to compare MPATCGA and MAGA. Since the major purpose of these experiments is to evaluate the optimization precision of MPATCGA, the experiments were implemented based on single PC platform. Because the implementation of this algorithm is based on single CPU, the parallel mechanism was changed as order parallel mechanism. Within one generation, each sub-population evolves in sequence. So the corresponding experimental process can be looked as fake parallel mode. The fake parallel mode can simplify the experimental process and is easy to be implemented by any interested authors no matter how many PCs he/she has. Theoretically, for optimization precision, the fake parallel mode is same as the true parallel mode, so the following experiments can show the optimization precision of MPATCGA and MAGA truly.

For the MPATCGA, The size of sub-population and the number of shared agents are adjustable. For fixed size of whole population, with the size of sub-population and the number of shared agents changes, the number of sub-population changes. The Table 2 shows the possible number of sub-populations with different shared agents and size of sub-population. Here, we discuss the number of sub-populations, size of sub-population, and number of shared agents based on one premise. The premise is that the whole population is same (however, the size of whole population in the experiments in this paper varies from 63 to 66 because the number of sub-populations should be integer).

In the following experiments in Sections 3.1.1–3.1.3, we set 6 as the size of sub-population and 2 as the number of shared agents. The setup of the other parameters is as follows: the size of whole population is 66, the probability of crossover is $p_c=0.95$, the probability of mutation is $p_m=0.05$, the upper limit of evolution
3.1. Low and middle dimensional optimization experiments of MPATCGA

We used MPATCGA and MAGA to optimize the tested functions listed in Table 1, respectively, under 2 dimensions, the statistical results were obtained after 50 running times and are listed in Table 3. The corresponding performance indices are as follows: ‘Fave’ means average global optima, ‘Generation’ means the average number of generation when optimization stops, ‘Time’ means average running time, ‘Time_Avg’ means average running time for each sub-population or for each CPU (here, the Time_Avg is obtained by the formula: $\text{Time}_\text{Avg} = \sum_{i=1}^{n} \text{Time}_{i}/k_{n\_sub}$, where ‘Time’ means the time cost of whole population with MPATCGA, $k_{n\_sub}$ means number of sub-populations).

From Table 3, we can find that under middle dimensions, the optimization of these functions is not very hard, so both the MPATCGA and MAGA can find similar optimization precision. For some test functions, the average running time of MPATCGA is shorter than that of MAGA. These advantages of MPATCGA are mainly in that: firstly, the whole population is divided into several sub-populations, the genetic operation becomes simpler, the relevant time is shortened. Secondly, within each sub-population, the number of neighboring individuals is changed from 4 to 2, thereby reducing the probability of some individuals with high fitness value occupying the whole population over early becomes less, that means the average optimization precision will become higher. Thirdly, the ring-like agent structure allows the genetic information propagate along the ring, which means for each sub-population, all the individuals in the sub-population communicate with other individuals. Just with some individuals, the whole sub-population can exchange genetic information with other sub-populations; the relevant genetic operation becomes less. Besides, if multi-CPU is used for realizing MPATCGA, the time cost can be reduced greatly.

3.1.1. Low and middle dimensional optimization experiments of MPATCGA

The following experiments will increase to 100 dimensions (see Table 4). The experimental conditions are similar as the low and middle dimensional optimization experiments (Section 3.1.1). With dimensions increasing, the coupling degree of variables becomes stronger, and the relevant optimization becomes more and more complex. The advantage of MPATCGA becomes more and more apparent. It means that for those high dimensional complex test functions, MPATCGA can obtain better optimization performance.

From the table, it can be seen that for high dimensional functions, MPATCGA has satisfied optimization precision. With dimensions increasing, the optimization precision falls down to some degree. It is because the increase of dimensions will lead to the increase of coupling degree of variables, at the same time, the search space will increase in positive proportion, approximately as $|I|^n$. However, MPATCGA still can find the more precise optimization results than MAGA. For example, for high dimensional multimodal functions $f_1$, $f_2$, $f_3$, the results can show that MPATCGA can have good global optimization capability for those high deceptive and multi-trap optimization problems. The ring-like (that is close chain-like) agent structure decreases the number of the neighboring individuals from 4 to 2, thereby reducing the probability of some individuals with high fitness value occupying the whole population over early, the diversity of population is kept. The structure is more effective for those functions with multi-local optima than lattice-like agent structure adopted in Zhong et al. (2004). Besides, the dynamic neighboring selection operator, neighboring crossover operator and adaptive mutation operator can contribute to the improvement. Similar to Section 3.1, if multi-CPU is used for realizing MPATCGA, the time cost can be reduced greatly.

3.1.2. High dimensional optimization experiments of MPATCGA

The following experiments will increase to 100 dimensions (see Table 4). The experimental conditions are similar as the low and middle dimensional optimization experiments (Section 3.1.1). With dimensions increasing, the coupling degree of variables becomes stronger, and the relevant optimization becomes more and more complex. The advantage of MPATCGA becomes more and more apparent. It means that for those high dimensional complex test functions, MPATCGA can obtain better optimization performance.

From the table, it can be seen that for high dimensional functions, MPATCGA has satisfied optimization precision. With dimensions increasing, the optimization precision falls down to some degree. It is because the increase of dimensions will lead to the increase of coupling degree of variables, at the same time, the search space will increase in positive proportion, approximately as $|I|^n$. However, MPATCGA still can find the more precise optimization results than MAGA. For example, for high dimensional multimodal functions $f_1$, $f_2$, $f_3$, the results can show that MPATCGA can have good global optimization capability for those high deceptive and multi-trap optimization problems. The ring-like (that is close chain-like) agent structure decreases the number of the neighboring individuals from 4 to 2, thereby reducing the probability of some individuals with high fitness value occupying the whole population over early, the diversity of population is kept. The structure is more effective for those functions with multi-local optima than lattice-like agent structure adopted in Zhong et al. (2004). Besides, the dynamic neighboring selection operator, neighboring crossover operator and adaptive mutation operator can contribute to the improvement. Similar to Section 3.1, if multi-CPU is used for realizing MPATCGA, the time cost can be reduced greatly.

3.1.3. Analysis of convergence performance

In order to show the convergence performance of MPATCGA and compare it with MAGA, the test functions listed in Table 1 were tested under several dimensions. Fig. 3 shows the convergence performance of MPATCGA and MAGA on function $f_4$, the experimental condition is similar to Sections 3.1.1 and 3.1.2. The function $f_4$ is complex high deceptive multimodal function, and very suitable for testing optimization performance. In Fig. 3, lattice means MAGA, cell means MPATCGA.

Table 3
The comparison of optimization performances of MPAGA and MAGA under 10 dimensions.

<table>
<thead>
<tr>
<th>Functions (10 dimensions)</th>
<th>Fave</th>
<th>Generation</th>
<th>Time (s)</th>
<th>Time_Avg (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 MPATCGA</td>
<td>-4142.4523</td>
<td>55.96</td>
<td>8.7537</td>
<td>8.7537</td>
</tr>
<tr>
<td>F1 MAGA</td>
<td>-4189.8288</td>
<td>26.26</td>
<td>5.1152</td>
<td>0.3197</td>
</tr>
<tr>
<td>F2 MPATCGA</td>
<td>2.8199</td>
<td>19.38</td>
<td>3.8778</td>
<td>3.8778</td>
</tr>
<tr>
<td>F2 MAGA</td>
<td>0</td>
<td>19.02</td>
<td>3.2012</td>
<td>0.2001</td>
</tr>
<tr>
<td>F3 MPATCGA</td>
<td>2.6645E-15</td>
<td>32.06</td>
<td>5.0166</td>
<td>5.0166</td>
</tr>
<tr>
<td>F3 MAGA</td>
<td>2.6645E-15</td>
<td>27.10</td>
<td>4.9408</td>
<td>0.3088</td>
</tr>
<tr>
<td>F4 MPATCGA</td>
<td>9.0790E-05</td>
<td>88.64</td>
<td>14.6815</td>
<td>14.6815</td>
</tr>
<tr>
<td>F4 MAGA</td>
<td>1.3125E-15</td>
<td>33.76</td>
<td>6.5163</td>
<td>0.4073</td>
</tr>
</tbody>
</table>

Table 4
The comparison of optimization performances of MPATCGA and MAGA under 100 dimensions.

<table>
<thead>
<tr>
<th>Function (100)</th>
<th>Fave</th>
<th>Generation</th>
<th>Time</th>
<th>Time_Avg (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 MPATCGA</td>
<td>-33803.1422</td>
<td>339.96</td>
<td>128.5109</td>
<td>128.5109</td>
</tr>
<tr>
<td>F1 MAGA</td>
<td>40808.4975</td>
<td>28.10</td>
<td>10.6249</td>
<td>0.6641</td>
</tr>
<tr>
<td>F2 MPATCGA</td>
<td>0.6903</td>
<td>145.40</td>
<td>61.2187</td>
<td>61.2187</td>
</tr>
<tr>
<td>F2 MAGA</td>
<td>19.26</td>
<td>6.5163</td>
<td>0.4073</td>
<td></td>
</tr>
<tr>
<td>F3 MPATCGA</td>
<td>1.5877</td>
<td>127.26</td>
<td>43.4392</td>
<td>43.4392</td>
</tr>
<tr>
<td>F3 MAGA</td>
<td>26.6654E-15</td>
<td>28.32</td>
<td>9.9617</td>
<td>0.6226</td>
</tr>
<tr>
<td>F4 MPATCGA</td>
<td>0.7911</td>
<td>356.60</td>
<td>158.0647</td>
<td>158.0647</td>
</tr>
</tbody>
</table>
Seen from Fig. 3, convergence curve can be divided into two parts: the fast falling part (part 1) that can be looked as global searching part and the slow falling part (part 2) that can be looked as local searching part. In part 1, more strongly the fitness value changes, the better the global searching capability of this algorithm will be. From the figure, under several dimensions, MPATCGA show its good convergence performance. Taking Fig. 3(d) as example, part 1 is within 10 generations, but part 1 of MAGA is extended to be 50 generations. The speed of falling of MPATCGA is faster than that of MAGA in part 1. It means MPATCGA can have better global searching capability, especially for those high deceptive multimodal functions. In part 2, the major task is to fix the optimal area. If the global searching capability of some algorithm is not strong, it is easy to fall into local trap for this algorithm. If the diversity of population can be kept well, it is easy to jump out of local trap to locate the global optima or near global optima. For function \( f_3 \), there are many local optima, which are very close, so the optimization of the function belongs to high deceptive problem. Seen from the experimental results, MPATCGA shows its good global searching capability. In part 2, good algorithm can tell different local optima within local searched area. As for MPATCGA, its neighboring selection operator and crossover operator keep the individuals different each other as much as possible, thereby effectively distinguish different
Note: TF: test function; NSA: number of shared agents; SSP: size of subpopulation; GEN: generation.

The study of the size of sub-population (f₁, f₂, f₃, f₄) under 100 dimensions.

<table>
<thead>
<tr>
<th>TF</th>
<th>NSA</th>
<th>SSP</th>
<th>Fave</th>
<th>GEN</th>
<th>Time (s)</th>
<th>Time_Avg (s)</th>
<th>TF</th>
<th>NSA</th>
<th>SSP</th>
<th>Fave</th>
<th>GEN</th>
<th>Time (s)</th>
<th>Time_Avg (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₁</td>
<td>1</td>
<td>4</td>
<td>-3972.6895</td>
<td>25.18</td>
<td>0.9406</td>
<td>0.0448</td>
<td>f₁</td>
<td>1</td>
<td>4</td>
<td>8.88E–16</td>
<td>25.54</td>
<td>1.0168</td>
<td>0.0484</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4189.8276</td>
<td>26.5</td>
<td>0.9408</td>
<td>0.0724</td>
<td></td>
<td></td>
<td></td>
<td>8.88E–16</td>
<td>25.44</td>
<td>0.9495</td>
<td>0.0730</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4089.7846</td>
<td>27.6</td>
<td>0.8520</td>
<td>0.0947</td>
<td></td>
<td></td>
<td></td>
<td>8.88E–16</td>
<td>26.28</td>
<td>0.8604</td>
<td>0.0956</td>
</tr>
<tr>
<td>f₂</td>
<td>1</td>
<td>4</td>
<td>-4189.8276</td>
<td>23.12</td>
<td>1.5539</td>
<td>0.0846</td>
<td>f₂</td>
<td>1</td>
<td>4</td>
<td>8.88E–16</td>
<td>25.12</td>
<td>1.4940</td>
<td>0.0467</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4189.8289</td>
<td>24.72</td>
<td>1.1449</td>
<td>0.0716</td>
<td></td>
<td></td>
<td></td>
<td>8.88E–16</td>
<td>25.72</td>
<td>0.7512</td>
<td>0.0469</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4166.0853</td>
<td>25.9</td>
<td>1.0831</td>
<td>0.0985</td>
<td></td>
<td></td>
<td></td>
<td>8.88E–16</td>
<td>25.7</td>
<td>1.0641</td>
<td>0.0967</td>
</tr>
<tr>
<td>f₃</td>
<td>1</td>
<td>4</td>
<td>-4189.8276</td>
<td>22.58</td>
<td>2.6354</td>
<td>0.0418</td>
<td>f₃</td>
<td>1</td>
<td>4</td>
<td>8.88E–16</td>
<td>24.82</td>
<td>2.9104</td>
<td>0.0462</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4189.8288</td>
<td>22.68</td>
<td>1.3599</td>
<td>0.0648</td>
<td></td>
<td></td>
<td></td>
<td>8.88E–16</td>
<td>25.22</td>
<td>1.4307</td>
<td>0.0681</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4166.1399</td>
<td>23.32</td>
<td>1.2119</td>
<td>0.0932</td>
<td></td>
<td></td>
<td></td>
<td>8.88E–16</td>
<td>25.7</td>
<td>1.3434</td>
<td>0.1033</td>
</tr>
</tbody>
</table>

Table 6

The study of the size of sub-population (f₁, f₂, f₃, f₄) under 100 dimensions.

<table>
<thead>
<tr>
<th>TF</th>
<th>NSA</th>
<th>SSP</th>
<th>Fave</th>
<th>GEN</th>
<th>Time (s)</th>
<th>Time_Avg (s)</th>
<th>TF</th>
<th>NSA</th>
<th>SSP</th>
<th>Fave</th>
<th>GEN</th>
<th>Time (s)</th>
<th>Time_Avg (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₁</td>
<td>1</td>
<td>4</td>
<td>38765.4302</td>
<td>23.34</td>
<td>13.56</td>
<td>0.61</td>
<td>f₁</td>
<td>1</td>
<td>4</td>
<td>0.0345</td>
<td>22.12</td>
<td>8.21</td>
<td>0.41</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>39579.9123</td>
<td>24.14</td>
<td>14.52</td>
<td>1.02</td>
<td></td>
<td></td>
<td></td>
<td>2.6645E–15</td>
<td>25.44</td>
<td>7.82</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>39988.9142</td>
<td>22.87</td>
<td>13.45</td>
<td>1.42</td>
<td></td>
<td></td>
<td></td>
<td>2.6645E–15</td>
<td>22.12</td>
<td>7.43</td>
<td>0.93</td>
</tr>
<tr>
<td>f₂</td>
<td>1</td>
<td>4</td>
<td>40882.2777</td>
<td>26.81</td>
<td>15.45</td>
<td>0.52</td>
<td>f₂</td>
<td>1</td>
<td>4</td>
<td>0.01234</td>
<td>23.58</td>
<td>8.11</td>
<td>0.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>41898.2764</td>
<td>29.63</td>
<td>9.45</td>
<td>0.59</td>
<td></td>
<td></td>
<td></td>
<td>2.6645E–15</td>
<td>25.43</td>
<td>7.23</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>41456.1324</td>
<td>28.45</td>
<td>14.32</td>
<td>1.42</td>
<td></td>
<td></td>
<td></td>
<td>2.6645E–15</td>
<td>22.92</td>
<td>11.91</td>
<td>1.14</td>
</tr>
<tr>
<td>f₃</td>
<td>1</td>
<td>4</td>
<td>42123.2321</td>
<td>29.12</td>
<td>44.23</td>
<td>0.74</td>
<td>f₃</td>
<td>1</td>
<td>4</td>
<td>0.0921</td>
<td>19.23</td>
<td>33.21</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>41898.2764</td>
<td>31.34</td>
<td>14.67</td>
<td>0.71</td>
<td></td>
<td></td>
<td></td>
<td>2.6645E–15</td>
<td>23.31</td>
<td>8.09</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>41899.4389</td>
<td>30.3</td>
<td>15.02</td>
<td>1.12</td>
<td></td>
<td></td>
<td></td>
<td>2.6645E–15</td>
<td>22.57</td>
<td>7.82</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Note: TF: test function; NSA: number of shared agents; SSP: size of subpopulation; GEN: generation.

The study of the number of shared agents and size of sub-populations

Through the experiments above, the optimization capability of MPATCGA has been verified. The major reason that sub-populations can co-evolve is that they can exchange genetic information each other through shared agents, so it is necessary to study the number of shared agents. In order to study the relationship between the number of shared agents and optimization capability, different numbers of shared agents are adopted; they are 1, 2, and 3. The size of sub-population is 6; the setup of the other parameters is similar to the experiments above. The test functions are f₁, f₂, f₃, f₄ (100 dimensions). The relevant data can be seen in Table 5. Time_Avg means the average running time for each CPU if the MPATCGA is realized by multi-CPUs in parallel.

From Table 5, it can be seen that with the number of shared agent increasing, the optimization result does not change apparently. Within the sub-population, the genetic information is propagated along ring, so any individual can obtain the information, so theoretically, one shared agent can exchange genetic information between different sub-populations. That is why different number of shared agent leads to similar optimization precision. Secondly, from the Time_Avg, we can find out, with the number of shared agents increasing, the time cost needed by each sub-population decreases. For example, for test function f₁, when number of shared agents increases from 1 to 2, the Time_Avg decreases from 0.6381 to 0.6284 s. When number of shared agents increases from 2 to 3, the Time_Avg decreases from 0.6284 to 0.5677 s. It is because the increase of shared agents can...
quicken the propagation of genetic information between sub-populations. However, from Table 2, we know that with number of shared agents increasing, number of sub-populations increases accordingly. It means more CPUs are needed, more computational resource is needed. Therefore, number of shared agents cannot be too many or too few; suitable number of shared agents should be made certain based on practical application.

Fig. 4 shows the convergence performance of MPATCGAs with different number of shared agents and compare them with MAGA. In the figure, lattice stands for MAGA, cellga-6-1 stands for MPATCGA whose size of sub-population is 6 and whose number of shared agents is 1, cellga-6-2 stands for MPATCGA whose size of sub-population is 6 and whose number of shared agents is 2, cellga-6-3 stands for MPATCGA whose size of sub-population is 6 and whose number of shared agents is 3.

From the figure, it can be seen: firstly, with the number of shared agent increasing, the optimization result does not change apparently. Both the three kinds of MPATCGAs can obtain the global optima or near global optima better than MAGA. Secondly, with number of shared agents increasing, the convergence speed quickens. In other words, in most cases, the convergence speed of the MPATCGA whose number of shared agents is 3 is fastest; it can reach the near global optima area most quickly. It is because the increase of shared agents can quicken the propagation of genetic information between sub-populations.

Tables 6 and 7 show the optimization performance with different number of sub-population under 10 and 100 dimensions. From the tables, we can see that, firstly, with size of sub-population increasing, the time cost needed by each sub-population increases. It is because the individuals within one sub-population increases, time complexity increases accordingly. Secondly, with size of sub-population increasing, the optimization precision does not change apparently. It is because for different size of sub-population, the size of whole population is same or similar. Thirdly, when the number of shared agents is fixed, with size of sub-population increasing, average time needed by each CPU increases, but number of CPUs decreases accordingly. Therefore, size of sub-population cannot be too big or too small; the suitable size of sub-populations should be made certain based on practical application.

### 3.1.5. Self-adaptability in global numerical optimization

In order to verify the self-adaptability of the MPATCGA in global numerical optimization, we organized two experiments. The 1st experiment is to verify the self-adaptability of the MPATCGA when the search space of some sub-population is bigger than other sub-populations very apparently. The 2nd experiment is to verify the self-adaptability of the MPATCGA when the search space of some sub-population is smaller than other sub-populations very apparently. Table 8 shows the results. In the table, case 1 represents the 1st experiment and case 2 represents the 2nd experiment. We use the test function 2 for optimization with 100 dimensions. In case 1, we divide the search space into \( n_i \) sub-search spaces, each sub-search space is for one sub-population for searching. But the 3rd sub-search space is very huge compared with other sub-search spaces. It means the workload on the 3rd sub-population is more than other sub-populations apparently. In case 2, we divide the search space into \( n_i \) sub-search spaces, each sub-search space is for one sub-population for searching. But the 4th sub-search space is very small compared with other sub-search spaces. It means the workload on the 4th sub-population is less than other sub-populations apparently.

From the table, we can see that in case 1, if the size of sub-population cannot change, the 3rd sub-population's size remains no change. For the huge workload, the presetting size of 3rd sub-population (number of agents) is not enough apparently, so the precision cannot be guaranteed. But with self-adaptability of MPATCGA, the 3rd sub-population's size can change accordingly, so the precision can be guaranteed. Seen from the table, with self-adaptability, the 3rd sub-population can obtain the global optima (0), but without self-adaptability, the 3rd sub-population just can obtain the local optima (0.7749). Although, with self-adaptability, the time cost will increase a little, the sacrifice of a little time cost is very worthy considering the apparent improvement on precision. The case 2 is an opposite case. For smaller workload than other sub-populations, the agents seem more redundant. Although more agents are good to more precision, too redundant population cannot obtain higher precision linearly. The time cost will increase rapidly. So it is necessary to remove the redundancy. Self-adaptability can be helpful for this. Seen from the table, with self-capability, the size of 4th sub-population shrinks accordingly, the time cost decreases accordingly (0.4370 < 0.6190 s). But the precision can be guaranteed, the best result is 0 (global optima).

### 3.2. Feature selection experiments

We know that the feature selection means the searching of the optimal features combination through optimization method. Therefore, good optimization algorithm is essential for feature selection method. Here, four genetic algorithms including AGA (Srinivas and Patnaik, 1994), MAGA (Zhong et al., 2004), SFGA (Dun-wei et al., 2002), and SGA (Michalewicz and Fogel, 2000) are adopted to be compared with MPATCGA. The reasons for choosing these genetic algorithms are as follows: firstly, SGA is a traditional genetic algorithm with elitism strategy, and it is often used in feature selection and performs well, so it is suitable to be compared with other improved genetic algorithms. Secondly, AGA is a representative adaptive genetic algorithm and can keep the diversity of population effectively, so some GAs used in feature selection is adaptive GAs. The comparison with it can show MPATCGA has more powerful searching capability, can keep the diversity of population more effectively and avoid premature convergence to get near-global optima. Thirdly, FSGA is another improved genetic algorithm with adaptive crossover operator. It can adaptively adjust its probability of crossover to keep the diversity of population, and performs well for optimization problems. Fourthly, MAGA is an improved genetic algorithm with lattice-like agent population structure proposed recently. In Zhong et al. (2004), the MAGA is described to perform better than some well-known algorithms such as OGA/Q, AEA, FEP, BGA, so the comparison with it can show MPATCGA better performance over those genetic algorithms.

<table>
<thead>
<tr>
<th></th>
<th>F2 100 dimensions</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Case 1</td>
<td>Case 2</td>
<td>Case 2</td>
</tr>
<tr>
<td></td>
<td>No change</td>
<td>With self-adaptability</td>
<td>No change</td>
</tr>
<tr>
<td>Precision</td>
<td>0.7749</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Time cost</td>
<td>0.6761</td>
<td>0.7445</td>
<td>0.6190</td>
</tr>
</tbody>
</table>

Table 8: Self-adaptability on imbalanced workload on different sub-populations (F2, 100 dimensions).
Here the stop criterion is $k > 10$ or maximum number of iteration is more than 1000.

### Table 9
Comparison of feature selection capability of five GAs.

<table>
<thead>
<tr>
<th>DS</th>
<th>CP</th>
<th>SGAE</th>
<th>AGA</th>
<th>SFGA</th>
<th>MAGA</th>
<th>MPATCGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letter</td>
<td>ANF</td>
<td>10.2 ± 3.6</td>
<td>9.9 ± 2.7</td>
<td>10.5 ± 3.9</td>
<td>11 ± 2.8</td>
<td>10.2 ± 2.1</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>92.75 ± 3.4</td>
<td>95 ± 2.4</td>
<td>95 ± 4.8</td>
<td>93.5 ± 3.1</td>
<td>98 ± 1.7</td>
</tr>
<tr>
<td>Wave</td>
<td>ANF</td>
<td>18.4 ± 4.8</td>
<td>21.3 ± 4.3</td>
<td>18.4 ± 4.4</td>
<td>15.5 ± 3.8</td>
<td>14.9 ± 3.6</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>80.25 ± 4.8</td>
<td>74.25 ± 5.4</td>
<td>80.25 ± 4.8</td>
<td>77 ± 2.8</td>
<td>82.75 ± 2.6</td>
</tr>
<tr>
<td>Sonar</td>
<td>ANF</td>
<td>26.1 ± 1.9</td>
<td>26.0 ± 2.1</td>
<td>26.3 ± 2.5</td>
<td>25.7 ± 2.3</td>
<td>25.6 ± 2.1</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>89.8 ± 3.2</td>
<td>91.2 ± 4.8</td>
<td>90.7 ± 4.9</td>
<td>92.3 ± 5.1</td>
<td>94.6 ± 3.4</td>
</tr>
</tbody>
</table>

DS means dataset, CP means compared parameters, ANF means average number of selected features, ABF means the average best fitness value, and ACA means average classification accuracy of selected feature subset.

### Table 10
Comparison of feature selection capability of GAs-based wrapper method.

<table>
<thead>
<tr>
<th>DS</th>
<th>CP</th>
<th>SGAE</th>
<th>AGA</th>
<th>SFGA</th>
<th>MAGA</th>
<th>MPATCGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letter</td>
<td>ANF</td>
<td>10.2 ± 3.4</td>
<td>10.4 ± 2.6</td>
<td>10.1 ± 4.1</td>
<td>11.5 ± 3.1</td>
<td>10 ± 2.7</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>93.45 ± 3.4</td>
<td>95.3 ± 2.4</td>
<td>95.7 ± 4.8</td>
<td>97.5 ± 3.1</td>
<td>98.1 ± 1.7</td>
</tr>
<tr>
<td>Letter-4</td>
<td>ANF</td>
<td>12.6 ± 3.5</td>
<td>11.5 ± 2.1</td>
<td>11.9 ± 2.5</td>
<td>10.8 ± 2.1</td>
<td>10.1 ± 2.3</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>87.34 ± 3.7</td>
<td>89.3 ± 2.5</td>
<td>86.7 ± 3.1</td>
<td>91.3 ± 2.1</td>
<td>92.3 ± 1.9</td>
</tr>
<tr>
<td>Wave</td>
<td>ANF</td>
<td>18.4 ± 4.4</td>
<td>21.3 ± 4.5</td>
<td>18.4 ± 3.7</td>
<td>15.5 ± 3.9</td>
<td>14.9 ± 3.7</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>81.65 ± 4.1</td>
<td>78.49 ± 3.9</td>
<td>84.41 ± 3.8</td>
<td>78.1 ± 2.3</td>
<td>89.25 ± 2.4</td>
</tr>
<tr>
<td>Sonar</td>
<td>ANF</td>
<td>26.4 ± 3.9</td>
<td>25.3 ± 4.1</td>
<td>27.1 ± 3.9</td>
<td>24.9 ± 4.8</td>
<td>24.2 ± 3.4</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>93.2 ± 3.1</td>
<td>95.6 ± 3.3</td>
<td>95.1 ± 4.3</td>
<td>96.3 ± 2.9</td>
<td>97.7 ± 2.7</td>
</tr>
<tr>
<td>WDBC</td>
<td>ANF</td>
<td>14.6 ± 3.6</td>
<td>13.7 ± 3.5</td>
<td>14.1 ± 3.7</td>
<td>12.5 ± 2.5</td>
<td>12.4 ± 2.4</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>95.15 ± 2.2</td>
<td>95.98 ± 2.3</td>
<td>95.12 ± 2.6</td>
<td>96.93 ± 2.7</td>
<td>97.38 ± 1.9</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>ANF</td>
<td>11.2 ± 2.1</td>
<td>10.8 ± 1.9</td>
<td>11.1 ± 2.3</td>
<td>9.8 ± 1.9</td>
<td>8.7 ± 2.1</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>91.76 ± 3.1</td>
<td>92.34 ± 3.5</td>
<td>92.32 ± 2.9</td>
<td>9.43 ± 2.3</td>
<td>95.6 ± 2.5</td>
</tr>
</tbody>
</table>

### Table 12
Comparison of feature selection capability of five GAs with different classifiers.

<table>
<thead>
<tr>
<th>DS</th>
<th>CP</th>
<th>MPATCGA (-BP)</th>
<th>MPATCGA (-RBF)</th>
<th>MPATCGA (-SVM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letter</td>
<td>ANF</td>
<td>10 ± 2.7</td>
<td>10.2 ± 2.5</td>
<td>10.1 ± 2.3</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>98.1 ± 1.7</td>
<td>96.7 ± 2.1</td>
<td>98.8 ± 1.3</td>
</tr>
<tr>
<td>Letter-4</td>
<td>ANF</td>
<td>10.3 ± 2.4</td>
<td>10.9 ± 2.1</td>
<td>10.1 ± 2.0</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>92.4 ± 2.3</td>
<td>91.7 ± 2.7</td>
<td>92.3 ± 2.1</td>
</tr>
<tr>
<td>Wave</td>
<td>ANF</td>
<td>14.9 ± 3.7</td>
<td>15.4 ± 3.9</td>
<td>14.4 ± 3.5</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>89.25 ± 2.4</td>
<td>88.6 ± 2.9</td>
<td>91.17 ± 2.6</td>
</tr>
<tr>
<td>Sonar</td>
<td>ANF</td>
<td>24.2 ± 3.4</td>
<td>24.8 ± 3.9</td>
<td>24.4 ± 2.3</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>97.7 ± 2.7</td>
<td>98.8 ± 2.9</td>
<td>98.9 ± 1.9</td>
</tr>
<tr>
<td>WDBC</td>
<td>ANF</td>
<td>12.5 ± 2.2</td>
<td>13.1 ± 2.4</td>
<td>12.4 ± 2.1</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>97.92 ± 1.9</td>
<td>97.44 ± 2.1</td>
<td>94.56 ± 2.3</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>ANF</td>
<td>8.3 ± 2.4</td>
<td>8.8 ± 1.9</td>
<td>8.5 ± 2.1</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>96.1 ± 2.1</td>
<td>95.6 ± 1.7</td>
<td>96.2 ± 1.8</td>
</tr>
</tbody>
</table>

Indirectly, this group of experiments is conducted to show the satisfying search capability of MPATCGA for feature selection.

The datasets are selected from popular international UCI database, including binary-class and multi-class datasets, please see Table 9. Some of them are not tried before. This algorithm in this paper is applied in the new datasets not tried before. For different groups of experiments, some information changes and is not involved in this table. The fitness value of a chromosome or selected feature subset is evaluated using some kind of classifier and 5-fold cross validation (5-fold CV).

Here one evaluation criterion is used, and its corresponding fitness function is made up of two parts here, it is $\text{fitness}=\text{discriminability}\text{-correlation}$. The fitness function below is adopted for feature selection here: fitness function (evaluation criterion): $\text{fitness} = \sum_{i=1}^{N}(S_p/S_w) - \text{corr}^2$, where, $N$ is the number of the features; $S_p$ means between-classes variance, $S_w=(m_1-m_2)^2$; $m_1$ means the first class specimens under some feature; $m_2$ means the
second class specimens under the same feature; \( S_p = (\sigma_{\text{class}}^2 + (\sigma_{\text{class}}^2)^2) \) corr2 is correlation between features selected, and is called as within-classes variance. Here, the corr2 is to calculate the correlation of the features matrices of the two classes. corr2 is initialized as 0. The first step is to calculate the first feature vector and the second feature vector within the first class \( p_{\text{corr1}} \), then calculate the correlation of the first feature vector and the second feature vector within the second class \( p_{\text{corr2}} \), after that, the correlation of the first feature and the second feature \( p_{\text{corr}} \) can be obtained with the formula: \( p_{\text{corr}} = (p_{\text{corr1}} + p_{\text{corr2}})/2 \). The \( p_{\text{corr}} \) is added to corr2. With the same processing, the correlation of the second feature and the third feature can be gotten and be added to the corr2. The processing lasted until the correlation of the \((N-1)\)th feature and the \(N\)th feature is obtained and is added to the corr2. At this time, the corr2 is obtained.

### 3.2.1. Feature selection experiments with filter methods

Table 10 lists statistical experimental results (average results) of the 10 times experiments based on the former three datasets. From the table, it can be seen that the average number of features from MPATCGA is less than that from MAGA and SFGA, the variance of the average number of features from MPATCGA is lowest. According to the fitness function of evaluation criterion involved, the average best fitness value from MPATCGA is highest and most stable. For letter dataset, during 10 times experiments, the best fitness value does not change; it means that MPATCGA can search for the optima stably. For wave dataset, the variance of best fitness value just is \( 0.06 \). For both the two datasets, the average classification accuracy obtained by MPATCGA is highest, and the variance of the average classification accuracy is lowest.

### 3.2.2. Feature selection experiments with wrapper methods

In order to evaluate the feature selection precision of MPATCGA for wrapper feature selection, the previous GAs are used for comparison. For justice, all of the GAs use same classifier-BP neural network. Table 11 lists the experimental results; the relevant data are statistical results after ten times experiments.

From Table 11, it can be seen that with wrapper method, all the five GAs obtain improved feature selection precision than filter version. In terms of ANF, the MPATCGA can obtain least number of selected features, and is very stable. In terms of classification accuracy, the MPATCGA can obtain the highest classification accuracy (i.e. feature selection precision or optimization precision). For the wave dataset, all the five GAs' classification accuracy falls down. However, the MPATCGA still can obtain the highest classification accuracy. The experimental results show that the MPATCGA not only can easily be used for wrapper feature selection, but also can have satisfying classification accuracy and number of features compared with some other popular GAs. It means MPATCGA has good optimization precision over these datasets. For the multi-class dataset (Letter-4), the results are similar.

### 3.2.3. Feature selection experiments with different classifiers

In order to evaluate the sensitivity of optimization performance of MPATCGA to different classifiers, three frequently used classifiers are used here for comparison; they are BP, RBF, and SVM. The MPATCGA uses the three classifiers for wrapper feature selection, respectively. Table 12 lists the experimental results; the relevant data are statistical results after ten times experiments.

From Table 12, some things can be found: regardless of different classifiers, the average number of features and average classification accuracy does not change a lot. It means the

### Table 13

The study of the number of shared agents and size of subpopulations.

<table>
<thead>
<tr>
<th>DS</th>
<th>NSA</th>
<th>SSP</th>
<th>ANF</th>
<th>ACA</th>
<th>DS</th>
<th>NSA</th>
<th>SSP</th>
<th>ANF</th>
<th>ACA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letter</td>
<td>1</td>
<td>4</td>
<td>10.3 ± 2.9</td>
<td>96.2 ± 2.3</td>
<td>Sonar</td>
<td>1</td>
<td>4</td>
<td>26.2 ± 3.9</td>
<td>91.8 ± 2.9</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td>10.4 ± 3.1</td>
<td>96.7 ± 2.7</td>
<td></td>
<td></td>
<td>6</td>
<td>26.8 ± 3.3</td>
<td>95.3 ± 3.1</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>11.2 ± 2.6</td>
<td>97.1 ± 2.4</td>
<td></td>
<td></td>
<td>8</td>
<td>25.1 ± 3.1</td>
<td>95.9 ± 2.7</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>10.7 ± 3.6</td>
<td>96.8 ± 2.9</td>
<td></td>
<td></td>
<td>2</td>
<td>26.1 ± 3.3</td>
<td>96.1 ± 4.1</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td>10.3 ± 2.8</td>
<td>98.1 ± 1.7</td>
<td></td>
<td></td>
<td>6</td>
<td>24.2 ± 3.4</td>
<td>97.7 ± 2.7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>10.9 ± 2.3</td>
<td>98.4 ± 1.9</td>
<td></td>
<td></td>
<td>8</td>
<td>24.7 ± 3.6</td>
<td>98.2 ± 2.6</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>11.3 ± 2.9</td>
<td>97.1 ± 2.8</td>
<td></td>
<td></td>
<td>3</td>
<td>27.2 ± 3.9</td>
<td>93.2 ± 3.8</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td>10.4 ± 2.1</td>
<td>98.2 ± 1.9</td>
<td></td>
<td></td>
<td>6</td>
<td>25.6 ± 2.9</td>
<td>96.1 ± 2.8</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>10.2 ± 3.4</td>
<td>98.9 ± 1.6</td>
<td></td>
<td></td>
<td>8</td>
<td>24.1 ± 2.4</td>
<td>98.8 ± 2.4</td>
</tr>
<tr>
<td>Wave</td>
<td>1</td>
<td>4</td>
<td>15.3 ± 3.9</td>
<td>83.14 ± 2.9</td>
<td></td>
<td></td>
<td>1</td>
<td>4</td>
<td>9.3 ± 3.4</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td>14.1 ± 3.5</td>
<td>86.41 ± 2.8</td>
<td></td>
<td></td>
<td>6</td>
<td>9.1 ± 3.2</td>
<td>94.3 ± 3.2</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>14.4 ± 2.9</td>
<td>86.71 ± 3.1</td>
<td></td>
<td></td>
<td>8</td>
<td>9.0 ± 3.5</td>
<td>94.5 ± 3.4</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>15.7 ± 4.1</td>
<td>86.54 ± 4.2</td>
<td></td>
<td></td>
<td>2</td>
<td>8.9 ± 2.2</td>
<td>95.1 ± 2.2</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td>14.9 ± 3.7</td>
<td>89.25 ± 2.4</td>
<td></td>
<td></td>
<td>6</td>
<td>8.7 ± 2.1</td>
<td>95.6 ± 2.5</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>14.7 ± 3.1</td>
<td>89.13 ± 2.9</td>
<td></td>
<td></td>
<td>8</td>
<td>8.8 ± 2.3</td>
<td>96.7 ± 2.2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>15.1 ± 3.6</td>
<td>85.75 ± 3.6</td>
<td></td>
<td></td>
<td>3</td>
<td>9.2 ± 2.5</td>
<td>93.4 ± 2.5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td>15.9 ± 3.9</td>
<td>89.75 ± 3.1</td>
<td></td>
<td></td>
<td>6</td>
<td>9.6 ± 3.1</td>
<td>96.2 ± 2.3</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td>14.4 ± 3.1</td>
<td>89.88 ± 2.3</td>
<td></td>
<td></td>
<td>8</td>
<td>8.4 ± 2.1</td>
<td>96.8 ± 2.4</td>
</tr>
</tbody>
</table>

NSA means number of shared agents; SSP means size of sub-population.

### Table 14

Comparison with parallel feature selection method.

<table>
<thead>
<tr>
<th>DS</th>
<th>CP</th>
<th>MPATCGA (-BP)</th>
<th>Method 1 (-BP)</th>
<th>Method 2 (-BP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letter</td>
<td>ANF</td>
<td>10 ± 2.7</td>
<td>12.1 ± 4.3</td>
<td>13.3 ± 4.8</td>
</tr>
<tr>
<td></td>
<td>ACA</td>
<td>98.1 ± 1.7</td>
<td>92.1 ± 3.1</td>
<td>91.7 ± 3.7</td>
</tr>
<tr>
<td>Letter</td>
<td>ANF</td>
<td>10.3 ± 2.4</td>
<td>13.5 ± 4.5</td>
<td>14.3 ± 4.8</td>
</tr>
<tr>
<td></td>
<td>ACA</td>
<td>92.4 ± 2.3</td>
<td>78.4 ± 5.6</td>
<td>81.5 ± 4.2</td>
</tr>
<tr>
<td>Wave</td>
<td>ANF</td>
<td>14.9 ± 3.7</td>
<td>17.7 ± 3.7</td>
<td>18.1 ± 4.5</td>
</tr>
<tr>
<td></td>
<td>ACA</td>
<td>89.3 ± 2.4</td>
<td>73.2 ± 5.2</td>
<td>78.2 ± 4.8</td>
</tr>
<tr>
<td>Sonar</td>
<td>ANF</td>
<td>24.2 ± 3.4</td>
<td>28.9 ± 4.4</td>
<td>27.6 ± 5.6</td>
</tr>
<tr>
<td></td>
<td>ACA</td>
<td>97.7 ± 2.7</td>
<td>89.1 ± 4.3</td>
<td>89.8 ± 4.7</td>
</tr>
<tr>
<td>WDBC</td>
<td>ANF</td>
<td>12.5 ± 2.2</td>
<td>13.7 ± 3.4</td>
<td>14.1 ± 4.1</td>
</tr>
<tr>
<td></td>
<td>ACA</td>
<td>97.92 ± 1.9</td>
<td>91.23 ± 3.8</td>
<td>90.51 ± 4.1</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>ANF</td>
<td>8.3 ± 2.4</td>
<td>12.1 ± 3.6</td>
<td>13.3 ± 4.3</td>
</tr>
<tr>
<td></td>
<td>ACA</td>
<td>96.1 ± 2.1</td>
<td>89.9 ± 4.7</td>
<td>88.7 ± 4.9</td>
</tr>
</tbody>
</table>
MPATCGA can have good optimization performance regardless of different category of classifiers. In the practical feature selection, according to the concrete characteristics and requirements of the feature selection problems, the designers will choose the suitable different classifier. Since the MPATCGA is not sensitive to category of classifiers, the MPATCGA is a good optimization algorithm for different feature selection problems. For the multi-class dataset, the results are similar.

### 3.2.4. The study of the number of shared agents and size of sub-populations

For the MPATCGA, the size of sub-population and the number of shared agents are adjustable. For fixed size of whole population, with the size of sub-population and the number of shared agents changes, the number of sub-population changes. Here, we discuss the number of sub-populations, size of sub-population, and number of shared agents based on one premise. The premise is that the whole population is same (however, the size of whole population in the experiments in this paper varies from 63 to 66 since the number of sub-populations should be integer). Similar as Section 3.1.4, here, the considered possible numbers of shared agents are 1, 2, and 3, respectively; the considered possible sizes of sub-population are 4, 6, and 8, respectively. The purpose of this group of experiments is to study the effect of different values of parameters on the feature selection performance of MPATCGA. Here, the classifier used is BP neural network. Table 13 lists the experimental results; the relevant data are statistical results after ten times experiments.

From Table 13, we can find: firstly, with the number of shared agents and size of subpopulation changes, the feature selection performance changes accordingly. Theoretically, different number of shared agents means different capability of exchange genetic information, different size of sub-population means different searching capability of global optima. Secondly, the changes are not too much. Theoretically, when number of shared agents is one, the shared agent still can exchange genetic information between sub-populations. Thirdly, with number of the shared agents increasing, the feature selection precision becomes better. This is maybe because more shared agents will speed up the propagation of genetic information between sub-populations. Fourthly, with size of sub-population becomes larger, the feature selection precision becomes better. This is maybe because larger size of sub-population has more agents than smaller size of sub-population; naturally, the sub-population with larger size can obtain higher precision possibly.

#### 3.2.5. Comparison with parallel feature selection method

Currently, some researchers study the parallel feature selection algorithm based on optimization algorithms such as GA. Some of them divide the feature space (search space) into several sub-feature spaces (Silva and Fred, 2007). For each sub-feature space, one classifier is assigned to it for feature selection. Here, the representative method is compared with MPATCGA. The relevant papers do not state how to divide the feature space in detail, so we list two kind of division; they correspond to two kinds of methods (methods 1 and 2). For the first one, it fixes the former several features to divide the feature space into several sub-spaces; here we suppose the number of fixed features is 3, so the number of feature subspace is $2^3=8$. For the second one, it fixes the latter several features to divide the feature space into several sub-spaces; here we suppose the number of fixed features is 3, so the number of feature subspace is $2^3=8$ too. Including MPATCGA, all the three feature selection method use the same classifier (BP neural network) for wrapper feature selection. Table 14 lists the experimental results; the relevant data are statistical results after ten times experiments.

Table 14 shows that this algorithm MPATCGA is better than other two methods apparently in terms of number of features and classification accuracy. In terms of number of features, according to the three datasets, the MPATCGA can obtain least features, and the variance is lowest. In terms of classification accuracy, according to the three datasets, the MPATCGA can obtain best classification accuracy, and the variance is lowest. The reason maybe is that the other two methods divide the feature space into several sub-feature spaces, the division is fixed and cannot changes. However, the distribution of local optima and complexity within the feature space is not even, so the number of local optima and complexity for each sub-feature space is quite different. As we know, the size of sub-population for each sub-feature space is fixed and equal, so for the sub-feature space with most complexity and local optima and global optima (in other words, different sub-feature space means different computational load or computational cost), the corresponding sub-population is hard to obtain satisfying precision. For MPATCGA, the case is quite different. Although MPATCGA search the optimal feature subset in parallel, it has shared agents connecting the different sub-populations. With the shared agents, the genetic information can spread between different sub-populations. If one sub-population faces the too complex feature space, the shared agents will help other agents in other sub-populations to cooperate with the agents in this sub-population to search for optimal feature subset. In other words, the shared agents can balance the different computational load of different sub-populations. For the multi-class dataset, the results are similar.

#### 3.2.6. Self-adaptability in feature selection

In order to verify the self-adaptability of the MPATCGA in feature selection, we organized two experiments. The 1st experiment is to verify the self-adaptability of the MPATCGA when the search space of some sub-population is bigger than other sub-populations very apparently. The 2nd experiment is to verify the self-adaptability of the MPATCGA when the search space of some sub-population is smaller than others; the time cost for case 1 means the time cost of 3rd sub-population rather than whole population; the time cost for case 2 means the time cost of 4th sub-population rather than whole population.
space of some sub-population is smaller than other sub-populations very apparently. Here, the ‘search space’ means feature space. Table 15 shows the results. In the table, case 1 represents the 1st experiment; case 2 represents the 2nd experiment. We use the wave dataset for the two experiments. In case 1, we divide the search space into \( n_1 \) sub-search-spaces, each sub-search-space is for one sub-population for searching. But the 3rd sub-search space is very huge compared with other sub-search-spaces. It means the workload on the 3rd sub-population is more than other sub-populations apparently. In case 2, we divide the search space into \( n_2 \) sub-search-spaces, each sub-search-spaces is for one sub-population for searching. But the 4th sub-search space is very small compared with other sub-search-spaces. It means the workload on the 4th sub-population is less than other sub-populations apparently.

From the table, we can see that in case 1, if the size of sub-population cannot change, the 3rd sub-population's size remains no change. For the huge workload, the presetting size of 3rd sub-population (number of agents) is not enough apparently, so the precision cannot be guaranteed. But with self-adaptability of MPATCGA, the 3rd sub-population's size can change accordingly, so the precision can be guaranteed. Seen from the table, with self-adaptability, the 3rd sub-population just can obtain the local optima (1.1478). Although, with self-adaptability, the time cost will increase a little, the sacrifice of a little time cost is very worthy considering the apparent improvement on precision. The case 2 is an opposite case. For smaller workload than other sub-populations, the agents seem more redundant. Although more agents are good to more precision, too redundant population cannot obtain higher precision linearly. The time cost will increase rapidly. So it is necessary to remove the redundancy. Self-adaptability can be helpful for this. Seen from the table, with self-capability, the size of 4th sub-population shrinks accordingly, the time cost decreases accordingly (2.9891 < 4.8971 s). But the precision can be guaranteed, the best result is 1.3789, the corresponding classification accuracy is 81.89%.

4. Conclusions

In order to improve the deficiency of GAs with single population, this paper proposed one multi-population co-genetic algorithm with double chain-like agents structure (MPATCGA) for parallel numerical optimization, thereby reducing optimization time cost and keeping high optimization precision. This paper described the process of the design of this algorithm, and organized several experiments to test the algorithm’s optimization performance based on global numerical optimization experiments and feature selection experiments. Apparently, MPATCGA realizes the parallel optimization with multi-populations, so it can shorten the optimization time cost without question, but the advantage of optimization precision of MPATCGA needs to be verified. The numerical optimization experimental results show that MPATCGA has higher optimization precision and shorter optimization time cost than MAGA averagely, especially for those complex multimodal optimization problems under high dimensions. The feature selection experimental results show that MPATCGA has higher optimization precision than some well known GAs averagely, is not affected by different classifiers, and can be used for parallel feature selection than other optimization algorithms. It means that MPATCGA is very suitable for complex optimization problems such as global numerical optimization, feature selection, and so on, with satisfying optimization precision and speed. Within experiments, the number of shared agents and size of sub-population have been studied carefully both for numerical optimization and feature selection problems. With self-adaptability, the MPATCGA can deal with different kinds of imbalanced optimization problems with satisfying precision and time cost. The future works are to perfect this algorithm further and use it in the practical applications such as decision support system, data mining, bioinformatics, and so on.

Acknowledgement

The authors want to thank the reviews' and the editor's comments and the support of Chongqing Natural Science Foundation (NO: CSTC, 2008BB2164, CSTC, 2008BB2322) and Innovation Ability Training Foundation of Chongqing University (CDCX018).

References


