Expert Systems with Applications 36 (2009) 11570–11581

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Research of multi-population agent genetic algorithm for feature selection

Yongming Li *, Sujuan Zhang, Xiaoping Zeng

College of Communication Engineering, Chongqing University, Shazheng Street, Chongqing City 400030, China

ARTICLE INFO

Keywords:
Multi-population
Double chain-like agent structure
Genetic algorithm
Feature selection
Parallel

ABSTRACT

Search algorithm is an essential part of feature selection algorithm. In this paper, through constructing double chain-like agent structure and with improved genetic operators, the authors propose one novel agent genetic algorithm-multi-population agent genetic algorithm (MPAGAFS) for feature selection. The double chain-like agent structure is more like local environment in real world, the introduction of this structure is good to keep the diversity of population. Moreover, the structure can help to construct multi-population agent GA, thereby realizing parallel searching for optimal feature subset. In order to evaluate the performance of MPAGAFS, several groups of experiments are conducted. The experimental results show that the MPAGAFS cannot only be used for serial feature selection but also for parallel feature selection with satisfying precision and number of features.

1. Introduction

Currently, new applications dealing with huge amounts of data have been developed, such as mining data and medical data processing. In this kind of applications, a large number of features are usually available. Since the number of the features is high, if all the features are used for classification, the huge computational cost is intolerable. Besides, there are interferences between features that will lead to low classification accuracy. Therefore, feature selection is especially important when one is required to deal with a large or even overwhelming feature set (Cho, Kim, & Jeong, 2008; Christelle, Robert, & Nicolas, 2008; Guyon & Elissee, 2003; Oh, Lee, & Moon, 2004; Shah & Andrew, 2004).

Optimal feature selection can be viewed as an optimization problem to search for optimal feature combination (feature subset). Therefore, the search algorithm is essential to construct good feature selection algorithm. However, with large number of features, the searched feature space is very complex and full of local optima. It is not easy for the search algorithm to search for an optimal or near-optimal solution (feature subset) in this kind of search space. Suppose the input data D has $n_{samples}$ specimens and $n_{features}$ $X = \{x_i, i = 1, \ldots, n_{features}\}$, and the target classification variable c, the feature selection problem is to find from the $n_{features}$-dimensional observation space, $R^{n_{features}}$, a subspace of m features, $R^{n_{features}}$, that "optimally" characterizes c. Given a condition defining the "optimal characterization", a search algorithm is needed to find the best subspace. Because the total number of subspaces is $2^{n_{features}}$, and the number of subspaces with dimensions no larger than $n_{features}$ is $\sum_{i=1}^{n_{features}} \binom{n_{features}}{i}$, it is difficult to search the feature subspace exhaustively.

Many heuristic algorithms have been proposed for finding near-optimal solutions (Guyon & Elissee, 2003). Among these algorithms, greedy strategies that incrementally generate feature subsets have been proposed. Since these algorithms do not take into account complex interactions among features, in most of the cases they lead to sub-optimal solutions.

Because of its implicit parallelism which is a result of the evolution and the hereditary-like process, GA has been demonstrated to be an effective search tool for finding near-optimal solutions in complex and nonlinear search spaces as is the case of feature selection problems. Therefore, they have been widely used to solve feature selection problems (Cho et al., 2008; Christelle et al., 2008; Kudo & Sklansky, 2000; Oh et al., 2004; Shah & Andrew, 2004). In addition, a number of comparative studies have showed the superiority of GA's in feature selection problems including large numbers of features (Kudo & Sklansky, 2000). However, the traditional GA is too dependent on initial population; the simple genetic operators are easy to lead to a decrease in diversity of whole population with genetic operation. With search space becoming more complex, the traditional GA is easy to converge over early and falls into local optima. In order to improve its performance, many researchers did many works on it, and proposed some modified GAs.

These improvements relate to genetic operators, population size, population structure, selection strategy, and so on (Gong, Sun, & Guo, 2002; Leung & Wang, 2001; Muhlenbein & Schlierkamp-vose, 1993; Pan & Kang, 1997; Renders & Flasse, 1996; Srinivas & Patnaik, 1994; Tsai, Liu, & Chou, 2004; Tu & Lu, 2004; Yao et al., 1999; Zbigniew & Fogel, 2000; Zhong, Liu, Xue, & Jiao, 2004). Leung and Wang proposed an improved

* Corresponding author. Tel.: +86 023 65103544.
E-mail address: lymcentor924924@gmail.com (Y. Li).

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.03.032

genetic -algorithm- the orthogonal genetic algorithm with quantization (Leung & Wang, 2001). The algorithm can not only find optimal or close-to-optimal solutions but also can give more robust and significantly better results than some other improved GAs. Among the improved crossover operators and mutation operators, adaptive crossover and mutation operators are welcomed mostly (Chen, McPhee, & Yeh, 2007; Muhlenbein & Schlierkamp-vose, 1993; Pan & Kang, 1997). That is because the adaptive crossover operator and the mutation operator can effectively keep the diversity of population, avoiding premature convergence. For the adaptive crossover operator, incorrect setting of probability of crossover will be modified with genetic evolution one generation after one generation. Some researchers proposed penalty function to keep the diversity of population, the effect is satisfied to some extent (Hu, Wang, & Guo, 2005). However, according to various applications, it is hard to set penalty function correctly. Incorrect setting of penalty function will make performance of genetic algorithm fall. Moreover, additional introduction of penalty function except crossover and mutation operations will lead to more computational cost. Some researchers proposed hybrid genetic algorithm combining genetic algorithm and local optimization method together in order to improve genetic algorithm (Hwang & He, 2006; Min, Lee, & Han, 2006; Oh et al., 2004; Sitarz, 2006). Some other researchers proposed a competitive selection strategy (Elaoud, Teghem, & Bouaziz, 2007). The strategy is a special selection, the individual sometimes is different from the individuals selected, inheriting the good genes of individuals selected. It is seen that the strategy is similar to the local optimization method to some extent because they all can search the local region. Normally, the local optimization method is introduced between crossover operation and mutation operation, so the competitive selection strategy can have less computational cost because it functions as selection operation and as local optimization at the same time. Besides, the improvement of the population structure is important. For conventional genetic algorithm, the individuals live together, are selected, crossover and mutate together. The apparent disadvantage is some individuals with high fitness value are very early to cover the whole population leading to premature convergence. Since the selection, crossover and mutation processes need to be done within the whole population, the fitness value of the whole population and the individuals need to be calculated, the computational cost is much high. Zhong et al. (2004) proposed a lattice-like agent population structure to solve the problem. The individuals are thought as agents living in a lattice-like agent population structure, they do genetic operations just with neighborhood agents. The relevant experimental results show that the structure can effectively keep the diversity of the population. However, it is seen that this kind of structure is a two-dimensional structure; the agent needs to do genetic operations with four neighborhood agents. If a one-dimensional structure is introduced, one agent just needs to do genetic operations with two neighborhood agents, the computational cost will be reduced, and premature convergence will be avoided better. In real world, the genetic operations always happen within one local environment (i.e. community); therefore, if there are several subpopulations standing for local environments, the genetic performance maybe better.

On the other hand, currently, feature selection problems always are large scale, apparently one processor is not enough for running feature selection to meet the requirement of time cost. Therefore, parallelism is needed to be introduced into the genetic algorithm of feature selection algorithm to speed it up. Currently, some researchers did works on parallel feature selection based on genetic algorithm (Chen, Lee, & Chang, 2007; Garcia Lopez, Garcia Torres, & Melian Batista, 2006; Hugo & Fred, 2007; Melab, Cahon, Talbi, & Duponchel, 2002; Punch, Goldman, & Pei, 1993; Teixeira de Souza, Matwin, & Japkowicz, 2006).

Nordine Melab proposed one kind of parallel wrapper feature selection based on genetic algorithm (Melab et al., 2002). But strictly, just the evaluation of individuals is about parallelism. The authors classify the individuals into several groups, each group of individuals are assigned to one classifier to evaluate their fitness value. But the rest of genetic operations are done within one population in one processor. Chih-Ming Chen proposed another kind of parallel feature selection based on genetic algorithm (Chen et al., 2007). The algorithm is similar to that by Nordine Melab, just evaluate the individuals in parallel. Punch et al. proposed a similar algorithm too (Punch et al., 1993). Although the computational cost of evaluation of individuals is too large compared with other operations (including selection, crossover, mutation and so on), when the scale of feature selection problems becomes very huge, the computational cost of the genetic operations except evaluation of individuals cannot be negligible. If the genetic operations can be done in parallel, the computational cost will be reduced apparently.

As in the above discussion, the feature selection problem is essentially an optimization problem. Currently, there are two realization modes for parallel optimization based on 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 that it is hard to know whether the decomposition is reasonable, usually, incorrect decomposition will lead to bad performance. Hugo Silva and Ana Fred proposed one parallel feature selection based on genetic algorithm (Hugo & Fred, 2007). They divided the whole feature space (i.e. search space) into several sub-feature spaces; each sub-feature space is searched by one GA. The experimental results show that the method can effectively reduce computational cost. However, the method has the following drawbacks: firstly, the correct division of whole feature space is difficult. Secondly, when the feature space is very complex, the distribution of local optima is not even. The feature space for each GA is not different, so the search load for each processor is not even, the efficiency of parallel processing is not high. Potter and De Jong (2000) proposed one collaboration co-genetic algorithm. They introduced a multi-population idea into the genetic algorithm to improve GA's optimization performance. The algorithm divides whole feature space (search space) into several sub-feature spaces and divides whole population into several independent subpopulations with fewer variables. Each subpopulation searches for one optimal solution of one sub-feature space independently. After that, the optimal solutions of each sub-feature space are combined together to construct a final solution (i.e. feature subset). However, it still has some drawbacks: firstly, the algorithm only considers collaboration between subpopulations but does not consider competition between subpopulations. Secondly, the division of population into subpopulations is very challenging; maybe the division needs enough relevant prior knowledge, so it is not convenient for practical application. Besides, the false or not precise division will lead to false or not precise optimization result. Su and Hou (2008) proposed another multi-population genetic algorithm for the parameter optimization problem. The GA has two subpopulations, each subpopulation optimizes different object functions and two optimization results within one generation are obtained. After that, the two results are put together to make an optimal full solution. The division of population is dependent on the object function. If the object function can be divided into two parts only, the population must be divided into two subpopulations and cannot be divided into more than two subpopulations. The reduction of time cost is very limited.

Another kind of realization mode is to design one multi-population genetic algorithm, every subpopulation searches for optimal full solution of whole search space in parallel. The method has the following advantages: firstly, it is not necessary to divide the

whole search space into several sub search spaces. Secondly, the algorithm can make use of multi-population to search for solution in parallel to reduce the computational cost. However, since it is not necessary to divide the search space, for the same search space, smaller population will lead to worse precision. Apparently, the size of subpopulation is smaller than that of whole population. Therefore, it is necessary to enhance the interaction between subpopulations to improve their cooperation for optimization.

Based on the above analysis, one novel genetic algorithm – multi-population agent genetic algorithm with double chain-like agent structure (close chain-like agent structure and cycle chain-like agent structure) is proposed here for search algorithm of feature selection algorithm. For convenience, the algorithm is called MPAGAFSFS. In this algorithm, inside every subpopulation, close chain-like agent structure is applied; the individuals within one subpopulation are connected with one close chain as agents. Every subpopulation is connected to each other with one cycle chain and shares some common agents (they are called shared agents). The subpopulations evolve themselves and cooperate and compete with each other. Besides, the genetic operators are improved, they include dynamic neighborhood competition selection operator, neighborhood orthogonal crossover operator and adaptive mutation operator to effectively keep and enhance the diversity of the subpopulation, which is good to search for global optima in complex and high dimensional search space. As seen from the algorithm, it is more similar to the evolution in real world, genetic information first happens in the neighbors of the current individuals, and then spreads within some community, and then spreads to other communities. The application of this algorithm into feature selection will improve the performance of feature selection algorithm (the experimental results can support this point). In terms of parallel feature selection, because of multi-population, this algorithm can realize parallel feature selection with multi-processors, with each subpopulation running in one processor. Because of shared agents, the subpopulations can exchange genetic information with each other to search for optimal solution. With them, the precision can be assured.

This paper is organized as follows: In Section 2, the process of the design of the MPAGAFSFS is described. In Section 3, the efficiency of the proposed MPAGAFSFS for feature selection is evaluated through groups of comparison experiments. Finally, in Section 4 some conclusions are made, and some future work is discussed.

2. Analysis and realization of algorithm

2.1. Agent structure

2.1.1. Close chain-like agent structure inside subpopulation

In the 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 $1 \times L_{size}$, where L_{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.

Definition. Assuming that the agent that is located at $(1, i)$ is represented as $L_{1,i}, i = 1, 2, \ldots, L_{size}$, the neighbors of $L_{1,i}$, $Neibors_{1,i}$ are defined as follows:

$$Neibors_{1,i} = \{L_{1,i_1}, L_{1,i_2}\} \tag{1}$$

where $i_1 = \begin{cases} i-1 & i \neq 1 \\ L_{size} & i = 1 \end{cases}$, $i_2 = \begin{cases} i+1 & i \neq L_{size} \\ 1 & i = L_{size} \end{cases}$. The agent ring can be described as the one shown in Fig. 1. Each circle represents an agent, the data in a circle represent its position in the ring, and the agent can interact with the left neighborhood one and the right neighborhood one.

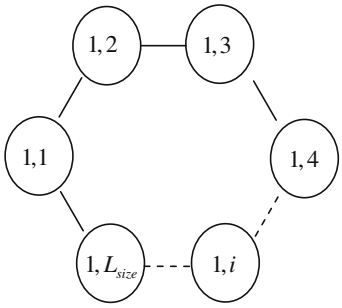

Fig. 1. Close chain-like agent structure inside subpopulation.

2.1.2. Multi-population cycle chain-like agent structure

Multi-population cycle chain-like agent structure means that in terms of the position information of agents, the whole population is divided into some subpopulations. The agents inside subpopulation are located in close chain-like agent structure and cooperate with each other. Each subpopulation is connected with other subpopulations 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 jth node in the ith subpopulation is expressed as $L_{i,j}$, where $i = 1, 2, \ldots, M, j = 1, 2, \ldots, L$. The neighborhood domain of $L_{i,j}$ is defined as: $Neibors_{i,j} = \{L_{i,j,n_1}, L_{i,j,n_2}\}$, where

$$L_{i,j,n_1} = \begin{cases} L_{i,L} & j = 1 \\ L_{i,j-1} & j \neq 1 \end{cases}, \quad L_{i,j,n_2} = \begin{cases} L_{i,1} & j = L \\ L_{i,j+1} & j \neq L \end{cases} \tag{2}$$

Fig. 2 shows the multi-population cycle chain-like agent structure with six agents per subpopulation and two 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 a new agent will occupy its position. Inside the subpopulation, 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 subpopulations, thereby improving the efficiency of the evolution.

2.2. Genetic operators

2.2.1. Dynamic neighborhood competition selection operator

The neighborhood competition selection operator is described as follows: suppose the order of competition selection is from left to right, the current agent is $L^t_{1,i}$, the neighbors are $Nbs_{1,i}$, $Nbs_{1,i} = \left\{L^t_{1,i1} \quad L^t_{1,i2}\right\}$, $i = 1, 2, \ldots\ldots, popsize$. Updating of $L^t_{1,i}$ is as the following formula:

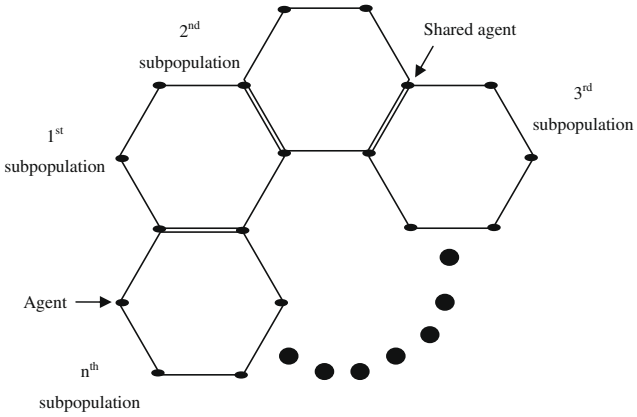

Fig. 2. Multi-population cycle chain-like agent structure.

$$\left\{\begin{array}{ll} L_{1,i}^t = L_{1,i}^t & fitness\left(L_{1,i}^t\right) > fitness(\max(L_{1,i1}, L_{1,i2})) \\ L_{1,i}^t = L_{1,i}^t \circ L_{1,i1}^t & \max(L_{1,i1}, L_{1,i2}) = L_{1,i1} \ \& \ fitness(L_{1,i1}) > fitness\left(L_{1,i}^t\right) \\ L_{1,i}^t = L_{1,i}^t \circ L_{1,i2}^t & \max(L_{1,i1}, L_{1,i2}) = L_{1,i2} \ \& \ fitness(L_{1,i2}) > fitness\left(L_{1,i}^t\right) \end{array}\right\} \tag{3}$$

In formula (3), $\circ$ indicates competition selection between agent $L_{1,i}^t$ and $L_{1,i1}^t$, the two agents consist of lots of genes:

$$\begin{aligned} L_{1,i}^t &= \left(c_{i,1}^t \quad c_{i,2}^t \quad \ldots \quad c_{i,j}^t \quad \ldots \quad c_{i,length}^t\right), \\ L_{1,i1}^t &= \left(c_{i1,1}^t \quad c_{i1,2}^t \quad \ldots \quad c_{i1,j}^t \quad \ldots \quad c_{i1,length}^t\right), \end{aligned} \tag{4}$$

$c_{i,j}^t$ indicates jth gene of $L_{1,i}^t$, $c_{i1,j}^t$ indicates jth gene of $L_{1,i1}^t$, *length* indicates number of genes of single agent. The competition selection between agent $L_{1,i}^t$ and $L_{1,i1}^t$ can be called $L_{1,i}^t \circ L_{1,i1}^t$, the processing is as follows:

$$\left\{\begin{array}{ll} c_{i,j}^t = c_{i,j}^t & c_{i,j}^t = c_{i1,j}^t \\ c_{i,j}^t = U(0,1) & c_{i,j}^t \neq c_{i1,j}^t \end{array}\right. \tag{5}$$

$U(0,1)$ indicates random number generator and is within the domain [0,1].

The procedures are as follows:

Step 1: define one register *temp* with space of (1×2), $temp \leftarrow \left(L_{1,i}^t, \quad \max(L_{1,i1}, L_{1,i2})\right)$;
Step 2: update $L_{1,i}^t$ according to formula (2);
Step 3: judge if $i = popsize$, if true, go to crossover processing, or not, $i \leftarrow i + 1$, turn to step 1.

Dynamic competition strategy: during the competition process, the $Max_{1,i} = \max(L_{1,i_1}, L_{1,i_2})$. The competition process is done in ascending order, after the competition of the 1st agent, the 1st agent is updated. The ith agents assumed before competition and after competition are $L_{1,i}^{pre}$ and $L_{1,i}^{post}$, respectively, so $Max_{1,i}$ is determined by

$$Max_{1,i} = \left\{\begin{array}{ll} \max\left(L_{1,L_{size}}^{pre}, L_{1,i+1}^{pre}\right) & i = 1 \\ \max\left(L_{1,L_{size}-1}^{post}, L_{1,1}^{post}\right) & i = L_{size} \\ \max\left(L_{1,i-1}^{post}, L_{1,i+1}^{pre}\right) & else \end{array}\right. \tag{6}$$

The updating process is a kind of computation process. From the principle of selection operator in this manuscript, we can know the competition between the current agent and its neighbors is not simple replacement. Through the competition (please see formula 3), a new agent is possibly created, that is why we say exploration and exploitation. The diversity is not destroyed but enhanced. For example, suppose dealing with ith agent, it is a, the previously updated $(i-1)$th agent is b, the later updated $(i+1)$th agent is c. If a competes with b, c is ignored. With the dynamic competition strategy, a competes with c, b is taken into account because c comes from b. Therefore, we think the agent based on the updated values of the neighbors rather than the original value will increase the diversity of population, thereby reducing the probability of early convergence.

2.2.2. Neighborhood adaptive crossover operator

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

$$p_{c,i} = \left\{\begin{array}{ll} \left(\frac{f_{\max} - f_i'}{f_{\max} - f_{ave}}\right)^{\frac{1}{GH(i,i')}} & f' \geqslant f_{ave} \\ 1 & f' < f_{ave} \end{array}\right. \tag{7}$$

Here, $p_{c,i}$ indicates the probability of crossover around the crossover operation between the $L_{1,i}$ and $Max_{1,i}$, $GH(i,i')$ indicates the distance between $L_{1,i}$ and $Max_{1,i}$, f' indicates the maximum value of both the individuals, $f_{\max}$ indicates the maximum value of all the individuals in the current population, f_{ave} indicates 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 $L_{1,i}$ and $Max_{1,i}$
else keep $L_{1,i}$ without change

2.2.3. Adaptive mutation operator

In the crossover process, the mutation probability p_m is calculated adaptively based on the length of the chromosome. The p_m is determined by $p_m = \frac{1}{n}$, where n indicates number of genes, namely, the length of the chromosome.

The crossover procedure is as follows:

if $U(0,1) < p_m$

do single point mutation processing between $L_{1,i}$ (namely, some gene changes its value from 1 to 0 or vice versa randomly)
else keep $L_{1,i}$ without change

2.3. Stopping criterion

f_{ave} can reflect the evolution of the current population. f_{best} indicates the best average fitness value since beginning. k_{stop} indicates a counter, it counts the number that f_{best} has no change. If $k_{stop} > k$, 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 equal 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 the experiments section.

2.5. Local search operator

Agents have knowledge which is related to the problems that they are designed to solve. According to our experiences, for optimization problems, integrating local searches with GAs can improve the performance. There are several ways to realize the local searches for feature selection (Hwang & He, 2006; Min et al., 2006; Oh et al., 2004; Sitarz, 2006) by combining GA and local search operator to obtain a satisfying performance. Enlightened by their idea, we propose the local search operator which uses a small scale MPAGAFS_IN (MPAGAFS_IN indicates MPAGAFSFS inside subpopulation, please see Section 2.6) to realize the behavior of using knowledge. The majority of the local searching does not include crossover operation. For more clarity, the algorithm about the local search operator is described as follows:

Step 1: $r \leftarrow 0$;
Step 2: do dynamic neighborhood competitive selection processing and update L^r, obtaining $L^{r+1/2}$;
Step 3: for each agent in $L^{r+1/2}$, do mutation processing on it, obtaining L_{end}^r;
Step 4: find ind_{best}^{cr} in L_{end}^r, and compare ind_{best}^{cr} and ind_{best}^{r-1}, if $Eng(ind_{best}^{cr}) > Eng\left(ind_{best}^{r-1}\right)$, then $ind_{best}^r \leftarrow ind_{best}^{cr}$, else, $ind_{best}^r \leftarrow ind_{best}^{r-1}$, $L^{r+1} \leftarrow L_{end}^r$;
Step 5: if stop criterion is satisfied, then go to the next generation of MPAGAFS_IN, else $i \leftarrow i + 1$, go to Step 2.

2.5.1. Comment

L^r represents the agent chain in the ith generation of local searching within some generation, and $L^{r+1/2}$ is the mid-chains between L^r and L^{r+1}, L^r_{end} is the agent chain after mutation processing in the ith generation of local searching within some generation. ind^r_{best} is the best agent since initialization of population and ind^{cr}_{best} is the best agent in L^r of local searching within some generation.

2.6. Realization of MPAGAFS algorithm

The MPAGAFS algorithm can be divided into two parts: MPAGAFSFS inside subpopulation (MPAGAFS_IN) and MPAGAFS between subpopulations (MPAGAFS_BETWEEN).

In MPAGAFS_BETWEEN, the major procedures are as follows:

Step 1: The initial population is obtained.
Step 2: The initial population is divided into M subpopulations with the size of L.
Step 3: Each subpopulation evolves, respectively.
Step 4: Judge whether all the subpopulations complete their one generation evolution, if so, the M best individuals are obtained; if not, continue step 3.
Step 5: The M best individuals can be judged and be used to update the best individual in the whole population in the current generation. At the same time, the evolution counter is updated.
Step 6: Judge whether the stop criterion is satisfied, if so, output the final best individual; if not, turn to step 3.

Suppose the best individual in the whole population in the ith generation is $ind^i_{best\_whole}$, the number of evolution counters is k_{cnt_whole}, the upper boundary of k_{cnt_whole} is TIMEs_OUT. The stopping criterion here is as follows: compare the $ind^i_{best\_whole}$ and $ind^{i-1}_{best\_whole}$, if the difference is lower than ε, k_{cnt_whole} is added with 1, or else, k_{cnt_whole} is updated with 0. When k_{cnt_whole} equals to TIMEs_OUT, quit the whole evolution and output the final optimization.

In MPAGAFS_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 neighborhood crossover operator and the mutation operator are applied on each agent, respectively. At the end of ith generation, the best agent in this generation competes with the best agent in $(i-1)$th generation, and ind^t_{best} (the best agent during t generations' evolution) is updated. The algorithm in this paper is described as follows:

Step 1: initialize L^0, update pop^0_{best}, and $t \leftarrow 0$;
Step 2: do dynamic neighborhood competitive selection processing and update L^t, obtaining $L^{t+1/3}$;
Step 3: for each agent in $L^{t+1/3}$, do crossover processing on it, obtaining $L^{t+2/3}$;
Step 4: for each agent in $L^{t+2/3}$, do mutation processing on it, obtaining L^t_{end};
Step 5: find ind^{ct}_{best} in L^t_{end}, and compare ind^{ct}_{best} and ind^{t-1}_{best}, if $Eng\left(ind^{ct}_{best}\right) > Eng\left(ind^{t-1}_{best}\right)$, then $ind^t_{best} \leftarrow ind^{ct}_{best}$, else, $ind^t_{best} \leftarrow ind^{t-1}_{best}$, $L^{t+1} \leftarrow L^t_{end}$;
Step 6: if stop criterion is satisfied, then output ind^t_{best} and stop, else $t \leftarrow t+1$, go to Step 2.

2.6.1. Comment

L^t represents the agent chain in the tth generation, and $L^{t+1/3}$ and $L^{t+2/3}$ are the mid-chains between L^t and L^{t+1}, L^t_{end} is the agent chain after mutation processing in the tth generation. ind^t_{best} is the best agent since initialization of population, and ind^{ct}_{best} is the best agent in L^t. p_c and p_m are the probabilities to perform the neighborhood crossover processing and the mutation processing.

Fig. 3 shows the block chart of MPAGAFS for feature selection.

As Fig. 3 shows, the MPAGAFS realizes the parallel feature selection. Different from some other parallel feature selection based on genetic algorithm, the genetic operations (such as selection, crossover, and mutation) are done in parallel. The computational cost is reduced further. At the same time, the subpopulations can co-operate with each other through shared agents to exchange genetic information in order to avoid reduction of precision because of smaller size of subpopulation. Besides, the genetic operators are improved with neighborhood competition selection operator, neighborhood adaptive operator and adaptive mutation operator, the precision of the algorithm is improved a lot. With this kind of parallelism, the search space (feature space) is not necessary to be divided. Usually, the wrong division of search space will lead to low efficiency of feature selection.

2.7. Analysis of computational complexity

In this section, we analyze the computational complexity of the proposed MPAGAFS. The time cost of ranking of individuals and crossover and mutation operation is smaller than the evaluation of test function, therefore, we define the computational cost of a single function evaluation as the basic unit of computational cost in our analysis.

For the simple GA, the computational complexity is $O(pg)$, here p is the size of the population and g is the number of search generations. For the MPAGAFS, since the MPAGAFS can realize the parallel feature selection based on genetic algorithm, the computational complexity should be $O(pg/n_n)$, suppose each subpopulation needs the same time cost. Here, n_n is the number of subpopulations. According to the theory of MPAGAFS, n_n can be calculated as follows: $n_n = \frac{N}{n_s - m}$, where, N indicates the size of whole population, m indicates the number of shared agents, n_s indicates the size of subpopulation. Therefore, the computational complexity of MAPGAFS should be $O(((n_s - m)pg)/N)$.

3. Experiments and analysis of results

From the block chart and computational complexity of MPAGAFS, it is seen that the algorithm can realize multi-subpopulation parallel feature selection, so the time cost can be reduced a lot. On comparison some experiments have shown the parallel feature selection can reduce the time cost and obtain the near linear speedup (Chen et al., 2007; Garcia Lopez et al., 2006; Hugo & Fred, 2007; Melab et al., 2002; Punch et al., 1993; Teixeira de Souza et al., 2006).

However, whether MPAGAFS can obtain the satisfying feature selection precision is not for sure. As we know, the feature selection precision is very important. If the precision becomes very bad, although the time cost can be reduced a lot, the algorithm still is not practical and meaningless. In order to show the feature selection precision of the MPAGAFS algorithm, several groups of experiments are conducted here. In the first group of experiments, several GAs including MPAGAFS are used for numerical optimization. The purpose of the experiments is to show the advantage of the agent structure of MPAGAFS. With this kind of agent structure, the precision of searching can be improved. In the second group of experiments, MPAGAFS is compared with several GA-based feature selection methods based on the same evaluation criterion. The purpose of the experiments is to show that MPAGAFS can be used for filter feature selection and gain good feature selection precision. In the third group of experiments, MPAGAFS is compared with several

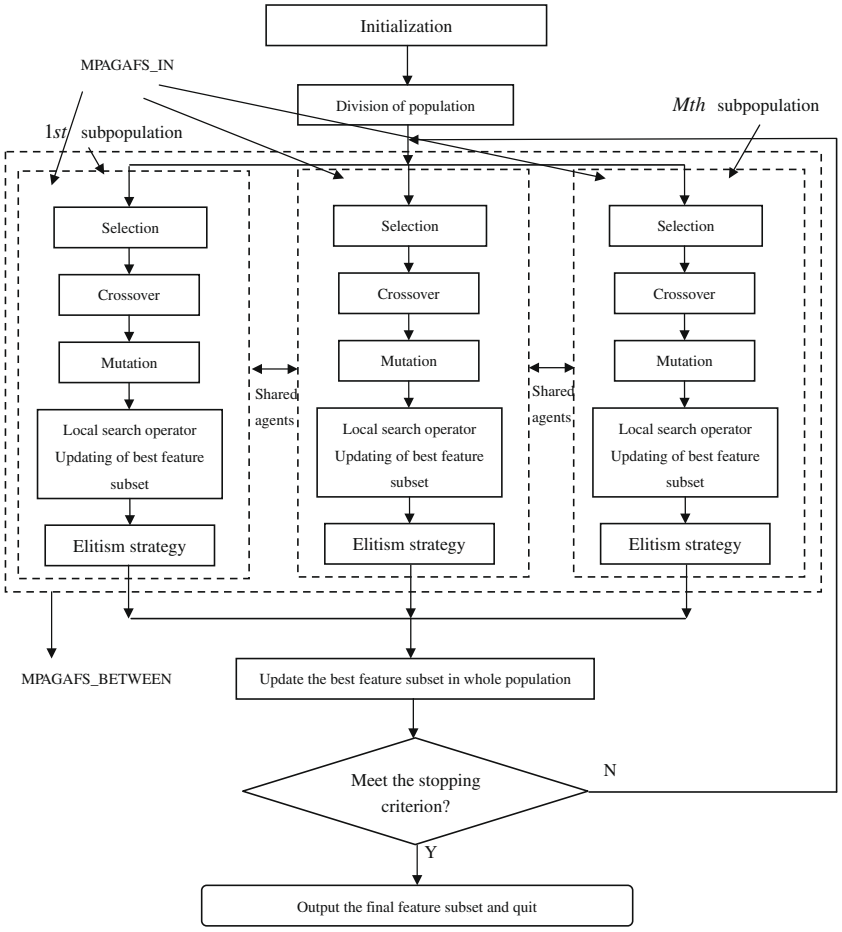

Fig. 3. The block chart of MPAGAFS for feature selection.

GA-based wrapper feature methods. The purpose of the experiments is to show that MPAGAFS can be used for wrapper feature selection and gain good feature selection precision. In the fourth group of experiments, MPAGAFS is used for feature selection based on different classifiers. The purpose of the experiments is to show that MPAGAFS can be used for wrapper feature selection method with different classifiers and gain satisfying feature selection precision. In the fifth group of experiments, different setups of key parameters of MPAGAFS are compared. The purpose of the experiments is to study the relationship between feature selection performance and the parameters. This study can help the designers to select suitable values of parameters according to concrete characteristics and requirements of feature selection problems. In the sixth group of experiments, MPAGAFS is compared with some other parallel feature selection methods. The purpose of the experiments is to show that MPAGAFS can be easily used for parallel feature selection and gain satisfying feature selection precision.

In the following experiments with Sections 3.1–3.4, 3.6, we set 6 as the size of subpopulation and 2 as the number of shared agents. The setup of the other parameters is as follows: the size of whole population is 66 approximately, the probability of crossover is $p_c = 0.95$, the probability of mutation is $P_m = 0.05$, the upper limit of evolution generation is $T = 1000$, TIMEs_OUT = 10. The relevant condition about PC platform is: CPU (central processing unit, CPU) with mainframe of 2.8 GHz, memory of 0.99 GB.

In the experiments in Section 3.5, the size of the whole population is 63–66; the size of the subpopulation and the number of shared agents are adjustable. The other parameters of MPAGAFS are the same.

3.1. Numerical optimization comparison experiments

As the previous discussion said, the feature selection problem is essentially an optimization problem. The search algorithm is opti-

Table 1
Test functions ($f_{\min}$ is the relevant global optima, n is dimensions of test functions, SD is searched domain).

Test functions	SD	$f_{\min}$
$f_1(x) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500, 500]^n$	$-418.95n$
$f_2(x) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	$[-5.12, 5.12]^n$	0
$f_3(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + e$	$[-32, 32]^n$	0
$f_4(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]^n$	0

mization algorithm to find the global optima (i.e. global optimal feature subset). Therefore, the group of numerical optimization experiments is organized here.

Some popular test functions were used in Table 1 for comparing MPAGAFS and MAGA (Zhong et al., 2004). $f_1 - f_4$ are multimodal functions with many local optima (traps).

Fig. 4 shows some test functions listed in Table 1. As the figure shows, function 1 and function 2 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 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 MPAGAFS and MAGA can show the advantage of the agent structure of MPAGAFS directly. Secondly, MAGA is an agent GA with single population, the comparison of MPAGAFS and MAGA can show that the MAPGA can not 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, and BGA, so the comparison with it can show MPAGAFS has better performance than those genetic algorithms indirectly.

3.1.1. Global numerical optimization experiments

Since the MAGA uses real code strategy, for justice, MPAGAFS uses real code strategy too. The corresponding selection operator, crossover operator and mutation operator are similar to that of MAGA for numerical optimization, for detail, please see Zeng, Li, and Qin (2009).

We used MPAGAFS and MAGA to optimize the test functions listed in Table 1, respectively, under 10 dimensions (see Table 2), 50 dimensions (see Table 3) and 100 dimensions (see Table 4), respectively, the statistical results were obtained after running 50 times. Since the emphasis on comparison is the optimization precision, both the two algorithms run in one processor, respectively. But we know, if giving multi-processors, MPAGAFS can be faster than MAGA. The corresponding performance indices are as follows: 'Fave' indicates average global optima, 'Generation' indicates the average number of generation when optimization stops, 'Time' indicates average running time, 'Time_Avg' indicates average running time for each subpopulation or for each CPU (here, the Time_Avg is obtained by the formula: Time_Avg = Time/k_{n_sub}, where 'Time' indicates the time cost of whole population with MPAGAFS, 'k_{n_sub}' indicates number of subpopulations.)

Table 2
Comparison of optimization performances of MPAGA and MAGA under 10 dimensions.

Functions (10 dimensions)		Fave	Generation	Time (s)
F1	MAGA	−4142.4523	55.96	8.7537
	MPAGAFS	−4189.8288	26.26	5.1152
F2	MAGA	2.8199	19.38	3.8778
	MPAGAFS	0	19.02	3.2012
F3	MAGA	2.6645E−15	32.06	5.0166
	MPAGAFS	2.6645E−15	27.10	4.9408
F4	MAGA	9.0790E−05	88.64	14.6815
	MPAGAFS	1.3125E−15	33.76	6.5163

Table 3
Comparison of optimization performances of MPAGA and MAGA under 50 dimensions.

Function (50 维)		Fave	Generation	Time (s)
F1	MAGA	−18242.8878	328.74	49.7018
	MPAGAFS	−20949.1382	27.82	6.0478
F2	MAGA	2.6436	102.46	16.0618
	MPAGAFS	0	19.22	4.2118
F3	MAGA	0.7002	108.68	16.7660
	MPAGAFS	2.6645E−15	27.22	5.0971
F4	MAGA	0.6658	270.70	42.2037
	MPAGAFS	0.3342	35.66	6.8683

From the tables, it is seen that for high dimensional functions, MPAGAFS has satisfied optimization precision. With increasing dimensions, the optimization precision reduces to some degree. It is because the increase in dimensions will lead to the increase in coupling degree of variables, at the same time, the search space will increase in positive proportion, approximately as $\prod_{i=1}^{n}(u_i - l_i)$. However, MPAGAFS still can find more precise optimization results than MAGA. For example, for high dimensional multimodal functions f_1, f_2, f_3, the results can show that MPAGAFS can have good global optimization capability for those high deceptive and multi-trap optimization problems. The close chain-like agent structure decreases the number of neighborhood individuals from 4 to 2, thereby reducing the probability of some individuals with high fitness value occupying the whole population early, the diversity of

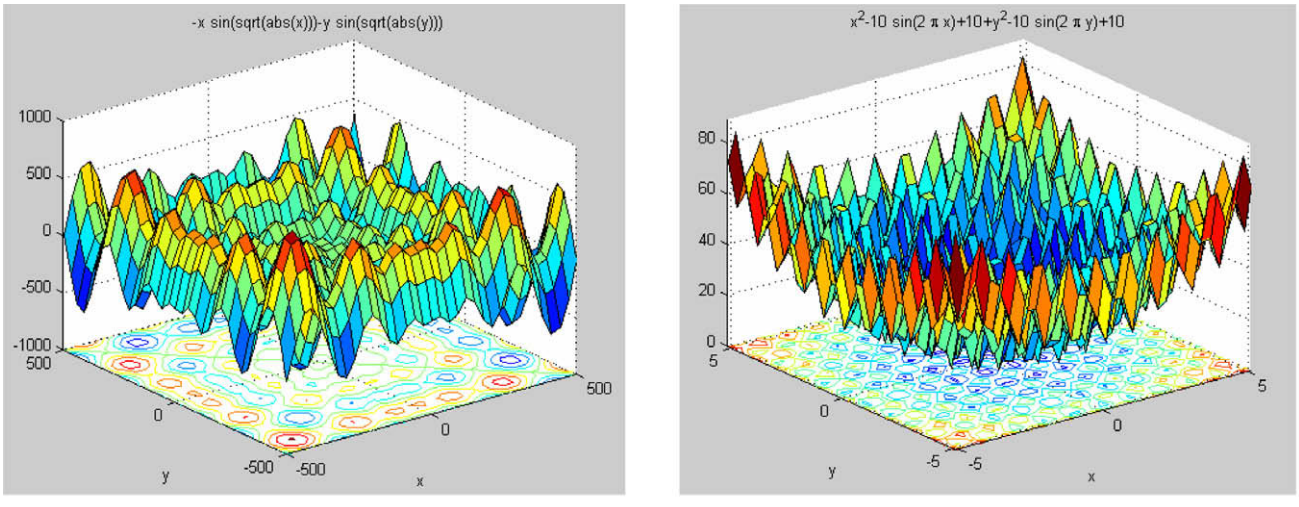

Fig. 4. Tested functions: (a) f_1 and (b) f_2.

Table 4
Comparison of optimization performances of MPAGA and MAGA under 100 dimensions.

Function (100 维)		Fave	Generation	Time
F1	MAGA	−33803.1422	339.96	12.8511
	MPAGAFS	−40608.4975	28.10	10.6249
F2	MAGA	0.6903	145.40	61.2187
	MPAGAFS	0	19.26	8.4375
F3	MAGA	1.5877	127.26	43.4392
	MPAGAFS	2.6645E−15	28.32	9.9617
F4	MAGA	0.7911	356.60	15.8065
	MPAGAFS	0.4087	48.54	13.7845

population is kept. The structure is more effective for those functions with multi-local optima than for lattice-like agent structure. Besides, the dynamic neighborhood selection operator, neighborhood crossover operator and adaptive mutation operator can contribute to the improvement. Even running in one processor, the MPAGAFS can obtain lower time cost than MAGA. With increase in dimensions, the advantage of time cost becomes more apparent. It means that for complex feature space, MPAGAFS is very suitable for search algorithm for this kind of feature space.

3.1.2. Analysis of convergence performance

In order to show the convergence performance of MPAGAFS and compare it with MAGA, several test functions listed in Table 1 were tested under several dimensions. Fig. 5 shows the convergence performance of MPAGAFS and MAGA for function f_2. Function f_2 is a complex high deceptive multimodal function, and very suitable for testing optimization performance. In Fig. 5, 'lattice' means MAGA, 'cell' means MPAGAFS.

From Fig. 5, 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 2, the diversity of population is very strong; the major task is to fix the optimal area. If the global searching capability of some algorithm is not strong, it is easy for this algorithm to fall into a local trap. If the diversity of population can be well kept, it is easy to jump out of local trap to locate the global optima or near-global optima. In part 1,the more strongly the fitness value changes, the better the global searching capability of this algorithm will be. From the figure, under several dimensions, MPAGAFS shows its good convergence performance. Taking Fig. 5d as an example, part 1 is within five generations, but the part 1 of MAGA is extended to 50 generations. The speed of converging of MPAGAFS is faster than that of MAGA in part1. It means MPAGAFS can have better global searching capability, especially for those high deceptive multimodal functions. For function f_2, there are many local optima, and the local optima are very close, so the optimization of the function belongs to high deceptive problem. From the experimental results, MPAGAFS shows its good global searching capability. Part 2 can be looked as local searching part, good algorithm can tell different local optima within local searched area. For MPAGAFS, its neighborhood selection operator and crossover operator keep the individuals different from each other as much as possible, thereby effectively distinguishing different optima. Besides, the adaptive mutation operator can reduce the search area gradually, thereby improving the searching precision and saving searching time. For f_2 under 100 dimensions, part 2 of MPAGAFS is 15 generations (from 5th generation to 20th generation), but part 2 of MAGA is 45 generations (from 50th generation to 95th generation). It means MPAGAFS has better local searching capability than MAGA.

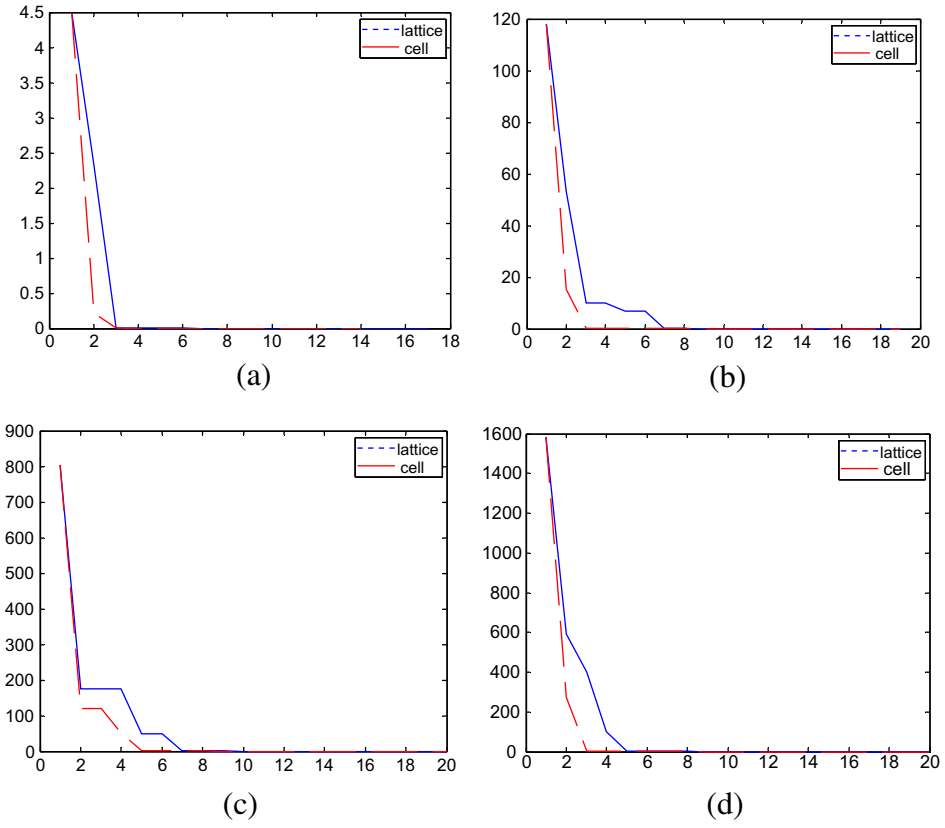

Fig. 5. Comparison of convergence performance of MPAGAFS and MAGA under different dimensions (a) 2 dimensions; (b) 10 dimensions; (c) 50 dimensions and (d) 100 dimensions.

3.2. Feature selection experiments with filter methods

We know that the feature selection means the search for the optimal feature combination through the optimization method. Therefore, good search algorithm is essential for the feature selection method. Here, four genetic algorithms including AGA (Srinivas & Patnaik, 1994), MAGA (Zhong et al., 2004), SFGA (Gong et al., 2002) and SGAE (Zbigniew & Fogel, 2000) are adopted to be compared with MPAGAFS. The reasons for choosing these genetic algorithms are firstly, SGAE 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 representational adaptive genetic algorithm and can effectively keep the diversity of population, so some GAs used in feature selection are adaptive GAs. The comparison with it can shows MPAGAFS 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 it performs well for optimization problems. Fourthly, MAGA is an improved genetic algorithm with the lattice-like agent population structure proposed recently. In Zhong et al. (2004), MAGA is described to perform better than some well-known algorithms such as OGA/Q, AEA, FEP, and BGA, so the comparison with it can show MPAGAFS has better performance over those genetic algorithms indirectly. This group of experiments is conducted to show the satisfying search capability of MPAGAFS for feature selection.

The datasets are selected from popular international UCI database. Dataset 1 is letter-recognition dataset; the number of features is 16. The classes *a* and *b* are used, there are 789 specimens belonging to class *a* and 766 specimens belonging to class *b*. Dataset 2 is a waveform dataset; the number of features is 40. The class wave 0 and wave 1 are used, there are 2000 specimens belonging to class wave 0 and wave 1, respectively. Dataset 3 is a Sonar dataset. There are 208 specimens belonging to two classes; the number of features is 60. For clarity, Table 5 shows some information about the experiments. Information on different groups of experiments differs and is not given in this table. The fitness value of a chromosome or selected feature subset is evaluated using some kind of classifier and fivefold cross-validation (fivefold CV).

Here one evaluation criterion is used, and its corresponding fitness function is made up of two parts here, it is: *fitness* = discriminability − correlation. The fitness function given below is adopted for feature selection here: Fitness function (evaluation criterion): $fitness = \sum_{i=1}^{N}(S_b/S_w)_i - corr2$, where N is the number of features; S_b indicates between-classes variance, $S_b = (m_1 - m_2)^2$; m_1 indicates first class specimens under some feature; m_2 indicates second class specimens under the same feature; $S_w = (\sigma_{class1})^2 + (\sigma_{class2})^2$; *corr2* is correlation between features selected, and is called as within-classes variance. Here, *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 correlation of the first feature vector and the second feature vector within the first class p_corr1, then to calculate the correlation of the first feature vector and the second feature vector within the second class p_corr2, after that, the correlation of the first feature and the second feature p_corr can be obtained from the formula: p_corr = (p_corr1 + p_corr2)/2. The p_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 *corr2*. The processing lasted until the correlation of the $(N-1)$th feature and the Nth feature is obtained and is added to the *corr2*. At this time, the *corr2* is obtained.

Table 6 lists statistical experimental results (average results) of experiments performed 10 times based on the former two datasets.

From Table 6, it can be seen that the average number of features from MPAGAFS is less than that from MAGA and SFGA, the variance of the average number of features from MPAGAFS is lowest. According to the fitness function of evaluation criterion involved, the average best fitness value from MPAGAFS is highest and most stable. For the letter dataset, for the experiment performed 10 times, the best fitness value does not change; it means that MPAGAFS can search for the optima stably. For the wave dataset, the variance of best fitness value is just ±0.06. For both the two datasets, the average classification accuracy obtained by MPAGAFS is highest, and the variance of the average classification accuracy is lowest.

3.3. Feature selection experiments with wrapper methods

In order to evaluate the feature selection capability of MPAGAFS for wrapper feature selection, the previous GAs are used for comparison. Since the other four GAs have no multi-population version, they have to run in one processor. Now that the comparison is emphasized on the precision, the MPAGAFS runs in one processor too, and the experimental results are still meaningful. For justice, all of the GAs use the same classifier-BP neural network. Table

Table 5
Some information about datasets.

Datasets	Number of features	Number of specimens	Number of classes	Population size	Evaluation of feature subset	Stop criterion
Letter	16	1555	2	30	5-fold CV	$k > 10$ or 1000
Wave	40	2000	2	30	5-fold CV	$k > 10$ or 1000
Sonar	60	208	2	30	5-fold CV	$k > 10$ or 1000

*Here the stop criterion is $k > 10$ or maximum number of iteration is more than 1000.

Table 6
Comparison of feature selection capability of five GAs.

DS	CP	SGAE	AGA	SFGA	MAGA	MPAGAFS
Letter	ANF	10.2, ±3.6	9.9, ±2.7	10.5, ±3.9	11, ±2.8	10.2, ±2.1
	ABF	17.6629, ±1.47	17.6897, ±0.23	17.9449, ±0.34	17.7852, ±0.12	17.9449, ±0
	ACA	92.75, ±3.4	95, ±2.4	95, ±4.8	93.5, ±3.1	98, ±1.7
Wave	ANF	18.4, ±4.8	21.3, ±4.3	18.4, ±4.4	15.5, ±3.8	14.9, ±3.6
	ABF	−0.1269, ±0.13	−0.5034, ±0.15	0.4300, ±0.11	1.5769, ±0.09	1.5467, ±0.06
	ACA	80.25, ±4.8	74.25, ±5.4	80.25, ±4.8	77, ±2.8	82.75, ±2.6

*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 7
Comparison of feature selection capability of GAs-based wrapper method.

DS	CP	SGAE	AGA	SFGA	MAGA	MPAGAFS
Letter	ANF	10.2, ±3.4	10.4, ±2.6	10.1, ±4.1	11.5, ±3.1	10, ±2.7
	ACA	93.45, ±3.4	95.3, ±2.4	95.7, ±4.8	97.5, ±3.1	98.1, ±1.7
Wave	ANF	18.4, ±4.4	21.3, ±4.5	18.4, ±3.7	15.5, ±3.9	14.9, ±3.7
	ACA	81.65, ±4.1	78.49, ±3.9	84.41, ±3.8	78.1, ±2.3	89.25, ±2.4
Sonar	ANF	26.4, ±3.9	25.3, ±4.1	27.1, ±3.9	24.9, ±4.8	24.2, ±3.4
	ACA	93.2, ±3.1	95.6, ±3.3	95.1, ±4.3	96.3, ±2.9	97.7, ±2.7

7 lists the experimental results; the relevant data are statistical results of experiments performed 10 times experiments.

From Table 7, it can be seen that with the wrapper method, all the five GAs obtain improved feature selection precision than the filter version. In terms of ANF, MPAGAFS can obtain least number of selected features, and is very stable. In terms of classification accuracy, MPAGAFS can obtain the highest classification accuracy. For the wave dataset, all the five GAs' classification accuracy falls down. However, MPAGAFS can still obtain the highest classification accuracy. The experimental results show that MPAGAFS 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.

3.4. Feature selection experiments with different classifiers

In order to evaluate the sensitivity of MPAGAFS to different classifiers, three frequently used classifiers are used here for comparison; they are BP, RBF, and SVM. MPAGAFS uses the three classifiers for wrapper feature selection, respectively. Table 8 lists the experimental results; the relevant data are statistical results of experiments performed 10 times.

From Table 8, the following can be found: firstly, regardless of different classifiers, the average number of features and average classification accuracy do not change much. It means MPAGAFS is not too sensitive to the 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 MPAGAFS is not sensitive to the category of classifiers, MPAGAFS can be used for different feature selections. Secondly, with SVM, MPAGAFS can obtain the highest classification accuracy to a certain extent.

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

For MPAGAFS, the size of subpopulation and the number of shared agents are adjustable. For the fixed size of whole population, with the size of subpopulation and the number of shared agents changing, the number of subpopulations changes. Here, we discuss the number of subpopulations, size of subpopulation, and the number of shared agents based on one premise. The premise is that the whole population is the same (however, the size of whole population in the experiments in this paper varies from 63 to 66 because the number of subpopulations should be an integer.). Here, the considered possible numbers of shared agents are 1, 2, and 3 respectively; the considered possible sizes of subpopulations 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 MPAGAFS. Here, the classifier used is BP neural network. Table 9 lists the experimental results; the relevant data are statistical results of experiments performed 10 times.

From Table 9, we can find firstly, that with the number of shared agents and size of subpopulation changing, the feature selection performance changes accordingly. Theoretically, different number of shared agents means different capability of exchange genetic information, different size of subpopulation means different searching capability of global optima. Secondly, the changes are not too much. Theoretically, when the number of shared agents is one, the shared agent can still exchange genetic information between subpopulations. Thirdly, with the number of shared agents increasing, the feature selection precision becomes better. This maybe because more shared agents will speed up the propagation of genetic information between subpopulations. Fourthly, with the size of subpopulation becoming larger, the feature selection precision becomes better. This maybe because larger size of subpopulation has more agents than smaller size of subpopulation; naturally, the subpopulation with larger size can possibly obtain higher precision.

Table 8
Comparison of feature selection capability of five GAs with different classifiers.

DS	CP	MPAGAFS (-BP)	MPAGAFS (-RBF)	MPAGAFS (-SVM)
Letter	ANF	10, ±2.7	10.2, ±2.5	10.1, ±2.3
	ACA	98.1, ±1.7	96.7, ±2.1	98.8, ±1.3
Wave	ANF	14.9, ±3.7	15.4, ±3.9	14.4, ±3.5
	ACA	89.25, ±2.4	88.6, ±2.9	91.17, ±2.6
Sonar	ANF	24.2, ±3.4	24.8, ±3.9	24.4, ±2.3
	ACA	97.7, ±2.7	97.8, ±2.9	98.9, ±1.9

Table 9
Study of the number of shared agents and size of subpopulations.

DS	NSA	SSP	ANF	ACA
Letter	1	4	10.3, ±2.9	96.2, ±2.3
		6	10.4, ±3.1	96.7, ±2.7
		8	11.2, ±2.6	97.1, ±2.4
	2	4	10.7, ±3.6	96.8, ±2.9
		6	10.3, ±2.8	98.1, ±1.7
		8	10.9, ±2.3	98.4, ±1.9
	3	4	11.3, ±2.9	97.1, ±2.8
		6	10.4, ±2.1	98.2, ±1.9
		8	10.2, ±3.4	98.9, ±1.6
Wave	1	4	15.3, ±3.9	85.14, ±2.9
		6	14.1, ±3.5	86.41, ±2.8
		8	14.4, ±2.9	86.71, ±3.1
	2	4	15.7, ±4.1	86.54, ±4.2
		6	14.9, ±3.7	89.25, ±2.4
		8	14.7, ±3.1	89.13, ±2.9
	3	4	15.1, ±3.6	85.75, ±3.6
		6	15.9, ±3.9	89.75, ±3.1
		8	14.4, ±3.1	89.88, ±2.3
Sonar	1	4	26.2, ±3.9	91.8, ±2.9
		6	26.8, ±3.3	95.3, ±3.1
		8	25.1, ±3.1	95.9, ±2.7
	2	4	26.1, ±3.3	96.1, ±4.1
		6	24.2, ±3.4	97.7, ±2.7
		8	24.7, ±3.6	98.2, ±2.6
	3	4	27.2, ±3.9	93.2, ±3.8
		6	25.6, ±2.9	96.1, ±2.8
		8	24.1, ±2.4	98.8, ±2.4

*NSA means Number of shared agents; SSP means Size of subpopulation.

Table 10
Comparison with parallel feature selection method.

DS	CP	MPAGAFS (-BP)	Method 1 (-BP)	Method 2 (-BP)
Letter	ANF	10, ±2.7	12.1, ±4.3	13.3, ±4.8
	ACA	98.1, ±1.7	92.1, ±3.1	91.7, ±3.7
Wave	ANF	14.9, ±3.7	17.7, ±3.7	18.1, ±4.5
	ACA	89.3, ±2.4	73.2, ±5.2	78.2, ±4.8
Sonar	ANF	24.2, ±3.4	28.9, ±4.4	27.6, ±5.6
	ACA	97.7, ±2.7	89.1, ±4.3	89.8, ±4.7

3.6. Comparison with parallel feature selection method

Currently, some researchers study the parallel feature selection algorithm-based on GA. As the discussion in Section 1, some of them divide the feature space (search space) into several sub-feature spaces Hugo and Fred, 2007. For each sub-feature space, one classifier is assigned for feature selection. Here, the representative method is compared with MPAGAFS. The relevant papers do not state how to divide the feature space in detail, so we list two kinds of divisions; they correspond to two kinds of methods (method 1 and method 2). In the first one, the former features are fixed to divide the feature space into several subspaces; here we suppose that the number of fixed features is 3, so the number of feature subspaces is $2^3 = 8$. In the second one, the latter features are fixed to divide the feature space into several subspaces; here we suppose that the number of fixed features is 3, so the number of feature subspaces is $2^3 = 8$ too. Including MPAGAFS, all the three-feature selection methods use the same classifier (BP neural network) for wrapper feature selection. Table 10 lists the experimental results; the relevant data are statistical results of experiments performed 10 times.

Table 10 shows that this algorithm MPAGAFS is better than the other two methods apparently in terms of number of features and classification accuracy. In terms of number of features, according to the three datasets, MPAGAFS can obtain least features, and the variance is lowest. In terms of classification accuracy, according to the three datasets, MPAGAFS can obtain best classification accuracy, and the variance is lowest. The reason maybe that the other two methods divide the feature space into several sub-feature spaces, the division is fixed and cannot change. However, the distribution of local optima and complexity within the feature space is not even, so the number of local optima and complexities for each sub-feature space is quite different. As we know, the size of subpopulation for each sub-feature space is fixed and equal, so for the sub-feature space with most complexity and local optima and with global optima (in other words, different sub-feature space means different computational load or computational cost), it is difficult for the corresponding subpopulation to obtain satisfying precision. For MPAGAFS, the case is quite different. Although MPAGAFS searches the optimal feature subset in parallel, it has shared agents connecting the different subpopulations. With the shared agents, the genetic information can spread between different subpopulations. If one subpopulation faces the too complex feature space, the shared agents will help other agents in other subpopulations to cooperate with the agents in this subpopulation to search for optimal feature subset. In other words, the shared agents can balance the different computational load of different subpopulations.

4. Conclusions

Search algorithm is an essential part of feature selection algorithms. Looking for good search algorithm with higher precision and faster speed is what the designers care about most. In this paper, through constructing double chain-like agent structure and improved genetic operators, the authors propose one novel agent genetic algorithm-multi-population agent genetic algorithm (MPAGAFS) for feature selection. The double chain-like agent structure is more like local environment in real world, the introduction of this structure is good to keep the diversity of population. Moreover, the structure can help to construct multi-population agent GA, thereby realizing parallel searching for optimal feature subset. In order to evaluate the performance of MPAGAFS, several groups of experiments are conducted. The experimental results show that MPAGAFS cannot only be used for serial feature selection but also for parallel feature selection with satisfying precision and number of features.

Now that the paper proposes one improved GA, the authors compare it with some other popular GAs to evaluate its performance in terms of numerical optimization problems and feature selection problems. Recently, some other optimization algorithms are used for feature selection, such as relief, particle swarm optimization, and colony optimization. In the near future, these optimization methods will be considered to be compared with MPAGAFS.

Acknowledgements

We would like to express our sincere thanks for the valuable suggestions and comments made by the reviewers and the editor. This research is funded by Chongqing Natural Science Foundation (No. is CSTC, 2008BB2164, CSTC, 2008BB2322) and Innovation Ability Training Foundation of Chongqing University (CDCX018).

References

Chen, C.-M., Lee, H.-M., & Chang, Y.-J. (2007). Two novel feature selection approaches for web page classification. *Expert Systems with Applications*. Available online: 1, October 2007.

Chen, L., McPhee, J., & Yeh, W. W.-G. (2007). A diversified multiobjective GA for optimizing reservoir rule curves. *Advances in Water Resources, 30*(5), 1082–1093.

Cho, H.-W., Kim, S. B., Jeong, M. K., et al. (2008). Genetic algorithm-based feature selection in high-resolution NMR spectra. *Expert Systems with Applications, 35*(3), 967–975.

Christelle, R., Robert, S., Nicolas, M., et al. (2008). A new genetic algorithm in proteomics: Feature selection for SELDI-TOF data. *Computational Statistics and Data Analysis, 52*(9), 4380–4394.

Elaoud, S., Teghem, J., & Bouaziz, B. (2007). Genetic algorithms to solve the cover printing problem. *Computers & Operations Research* (pp. 1–16).

Garcia Lopez, F., Garcia Torres, M., & Melian Batista, B. (2006). Solving feature subset selection problem by a parallel scatter search. *European Journal of Operational Research, 169*(2), 477–489.

Gong, D.-W., Sun, X.-Y., & Guo, X.-J. (2002). Novel survival of the fittest genetic algorithm. *Control and Decision, 17*(6), 908–911.

Guyon, I., & Elissee, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research, 3*, 1157–1182.

Hu, Y.-B., Wang, Y.-P., & Guo, F.-Y. (2005). A new penalty based genetic algorithm for constrained optimization problems. In *Proceedings of FICMLG'05* (pp. 3025–3029).

Silva, H., & Fred, A. (2007). *Feature subspace ensembles: A parallel classifier combination scheme using feature selection*: LNCS (Vol. 4472, pp. 261–270).

Hwang, S.-F., & He, R.-S. (2006). Improving real-parameter genetic algorithm with simulated annealing for engineering problems. *Advances in Engineering Software, 37*(6), 406–418.

Kudo, M., & Sklansky, J. (2000). Comparison of algorithms that select features for pattern classifiers. *Pattern Recognition, 33*(1), 25–41.

Leung, Y. W., & Wang, Y. (2001). An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE Transactions on Evolutionary Computation, 5*(1), 41–53.

Melab, N., Cahon, S., Talbi, E.-G., & Duponchel, L. (2002). Parallel GA-based wrapper feature selection for spectroscopic data mining. In *Proceedings of the international parallel and distributed processing symposium (IPDPS.02)*: IEEE Press. (pp. 201–208).

Michalewicz, Z., & Fogel, D. B. (2000). *How to solve it: Modern heuristics*. Berlin Heidelberg: Springer-Verlag. pp. 83–234.

Min, S.-H., Lee, J., & Han, I. (2006). Hybrid genetic algorithms and support vector machines for bankruptcy prediction. *Expert Systems with Applications, 31*(3), 652–660.

Muhlenbein, H., & Schlierkamp-vose, D. (1993). Predictive models for the breeder genetic algorithm. *Evolutionary computation, 1*(1), 25–49.

Oh, I.-S., Lee, J.-S., & Moon, B.-R. (2004). Hybrid genetic algorithms for feature selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 26*(11), 1424–1437.

Pan, Z. J., & Kang, L. S. (1997). An adaptive evolutionary algorithm in simulated evolution and learning. In X. Yao, J. H. Kim, & T. Furuhashi (Eds.). *Lecture Notes in Artificial Intelligence* (Vol. 1285, pp. 27–34). Berlin,Germany: Springer-Verlag.

Potter, M. A., & De Jong, K. A. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *IEEE Transactions on Evolutionary Computation, 8*(1), 1–29.

Punch, W. F., Goldman, E. D., Pei, M., et al. (1993). Further research on feature selection and classification using genetic algorithms. *International Conference on Genetic Algorithm*, 557–564.

Renders, J.-M., & Flasse, S. P. (1996). Hybrid methods using genetic algorithms for global optimization. *IEEE Transactions on Systems, Man and Cybernetics-Part B: Cybernetics, 26*(2), 243–258.

Shah, S. C., & Andrew, K. (2004). Data mining and genetic algorithm based gene/SNP selection. *Artificial Intelligence in Medicine, 31*(3), 183–196.

Sitarz, S. (2006). Hybrid methods in multi-criteria dynamic programming. *Applied Mathematics and Computation, 180*(1), 38–45.

Srinivas, M., & Patnaik, L. M. (1994). Adaptive probabilities of crossover genetic in mutation and algorithms. *IEEE Transactions on Systems, Man and Cybernetics, 24*(4), 656–667.

Su, C.-H., & Hou, T.-H. (2008). Using multi-population intelligent genetic algorithm to find the pareto-optimal parameters for a nano-particle milling process. *Expert Systems with Applications, 34*(4), 2502–2510.

Teixeira de Souza, J., Matwin, S., & Japkowicz, N. (2006). Parallelizing Feature Selection. *Algorithmica, 45*, 433–456. doi:10.1007/s00453-006-1220-3.

Tsai, J.-T., Liu, T.-K., & Chou, J.-H. (2004). Hybrid Taguchi-genetic algorithm for global numerical optimization. *IEEE Transactions on Evolutionary Computation, 8*(4), 365–377.

Tu, Z., & Lu, Y. (2004). A robust stochastic genetic algorithm (StGA) for global numerical optimization. *IEEE Transactions on Evolutionary Computation, 8*(5), 456–470.

Zeng, X.-P., Li, Y.-M., & Qin, J. (2009). A dynamic chain-like agent genetic algorithm for global numerical optimization and feature selection. *Neurocomputing, 72*(4), 1214–1228.

Yao, X., Liu, Y., & Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation, 3*(2), 82–102.

Zhong, W., Liu, J., Xue, M., & Jiao, L. (2004). A multiagent genetic algorithm for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, 34*(2), 1128–1141.