A Multi-agent Genetic Algorithm for Big Optimization Problems

Yutong Zhang, Mingxing Zhou, Zhongzhou Jiang, Jing Liu*
Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education,
Xidian University, Xian 710071, China
*Corresponding author, email: neouma@163.com

***Abstract*—With the coming of big data age, the data usually present in a huge magnitude such as TB or more. These data contain both useful and useless information. Therefore, techniques which can effectively analyze these data are in urgent demand. In practice, dealing with Electroencephalographic (EEG) signals with Independent Component Analysis (ICA) approximates to a big optimization problem because it requires real-time, or at least automatic in dealing with signals. Thus, in the Optimization of Big Data 2015 Competition, the problem abstracted from dealing with EEG signals through ICA is modeled as a big optimization problem (BigOpt). Evolutionary optimization techniques have been successfully used in solving various optimization problems, and in the age of big data, they have attracted increasing attentions. Since the multi-agent genetic algorithm (MAGA) shows a good performance in solving large-scale problems, in this paper, based on the framework of MAGA, an MAGA is proposed for solving the big optimization problem, which is labeled as MAGA-BigOpt. In MAGA-BigOpt, the competition and self-learning operators are redesigned and combined with crossover and mutation operators to simulate the cooperation, competition, and learning behaviors of agents. Especially, in the self-learning operator, agents quickly find decreasing directions to improve itself with a heuristic strategy. In the experiments, the performance of MAGA-BigOpt is validated on the given benchmark problems from the Optimization of Big Data 2015 Competition, where both the data with and without noise are used. The results show that MAGA-BigOpt outperforms the baseline algorithm provided by the competition in both cases with lower computational costs.**

Keywords: Multi-agent Genetic Algorithm, Big Data, Big Optimization

I. Introduction

Big Data is a term for any collection of datasets which are too large or complex to be processed by traditional methods. A definition coming from Internet Data Center (IDC) presents as "Big data technologies describe a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling high velocity capture, discovery, and or analysis". New reports are released everyday which describes the extent to which Big Data is influencing how we do business. Market research firm IDC forecasts that the market for Big Data is expected to grow at a 27% compound annual growth rate through 2017 to reach $32.4 billion and 2018 to $41.52 billion in its report.

Dealing with Electroencephalographic (EEG) signals with Independent Component Analysis (ICA) approximates to a big optimization problem because it requires real-time, or at least automatic in dealing with signals. Thus, in the Optimization of Big Data 2015 Competition [1]–[4] (IEEE Congress on Evolutionary Computation), the problem abstracted from dealing with EEG signals through ICA is modeled as a big optimization problem (BigOpt).

The problem is trying to decompose the obtained EEG signal into two components, making one of the components similar to the original signals so that the useful information are kept as much as possible and the other one to remove as much artifacts as possible. The dimension of this problem, which is the multiple of the number of channels and length of the EEG signals, can be extremely large, so it is difficult to be solved by traditional evolutionary algorithms.

In our previous work, a multi-agent genetic algorithm (MAGA) was proposed for large-scale global numerical optimization [5]. MAGA integrated multi-agent systems with GAs, and can optimize functions with 10000 dimensions, which is the first GA that can solve functions with such high dimensions. It is also extended to solve constraint satisfaction problems and combinatorial optimization problems successfully [6], [7]. Since MAGA showed a good performance, based on the framework of MAGA, we propose an MAGA for solving the big optimization problem, labeled as MAGA-BigOpt. In MAGA-BigOpt, four genetic operators, especially an effective sub-gradient based self-learning operator, are designed to simulate the cooperation, competition, and learning behaviors of agents. The algorithms can rapidly find a very excellent solution. In the experiments, we can see MAGA-BigOpt can provide much better solutions than the benchmark algorithm in both cases with lower computational costs.

The rest of this paper is organized as follows. The big optimization problem is introduced in Section II. MAGA-BigOpt is introduced in Section III, followed by the experiments in Section IV. Finally, the conclusion is given in Section V.

II. Big Optimization Problems

The big optimization problems are proposed in the Optimization of Big Data 2015 Competition [1] (IEEE Congress on Evolutionary Computation), which is based on a number of interdependent time series forming the datasets. Assume the matrix X is of dimension $N \times M$, where N is the number of interdependent time series and M is the length of each time

978-1-4799-7492-4/15/$31.00

series. Let S be a matrix of $N \times M$, with N independent time series of length M, such that, given A, an $N \times N$ linear transformation matrix,

$$X = A \times S \tag{1}$$

The problem is to decompose the matrix S into two matrices: $S^{(1)}$ and $S^{(2)}$ with the same dimensionalities of S; that is, $S = S^{(1)} + S^{(2)}$ and $X = A \times S^{(1)} + A \times S^{(2)}$. Let C be the Pearson correlation coefficient between $A \times S^{(1)}$ and X, that is,

$$C = \frac{covar(X, A \times S^{(1)})}{\sigma(X) \times \sigma(A \times S^{(1)})} \tag{2}$$

where $covar(\cdot)$ is the covariance matrix, and $\sigma(\cdot)$ is the standard deviation.

The objective is to maximize the diagonal elements of C, while minimizing off-diagonal elements to zeros. At the same time, the distance between S and $S^{(1)}$ should be as minimum as possible, that is, $S^{(1)}$ needs to be as similar as possible as S. This definition generates two formulations for the problem, one as a single objective, while the other as a multi-objective problem. The Multi-objective optimization problem is defined as follows, given, X, A, and S, find $S^{(1)}$ which optimizes the following two objective functions:

Information loss
$$\min f_1 = \frac{1}{N \times M} \sum_{i,j} (S_{ij} - S^{(1)}_{ij})^2 \tag{3}$$

Residual
$$\min f_2 = \frac{1}{N^2 - N} \sum_{i,j \neq i} C^2_{ij} + \frac{1}{N} \sum_{i} (1 - C_{ii})^2 \tag{4}$$

The single objective formulation, studied in this paper, is defined as follows, given X, A, and S, find $S^{(1)}$ which optimizes the following objective function:

$$\min F = f_1 + f_2 \tag{5}$$

The number of time series together with the length of each time series defines the number of variables in the problem. For this competition, each time series is of length 256; thus, each optimization problem has a multiple of 256 variables. For example, if the dataset has 4 time series, this will define a 1024 variables optimization problem. In this year competition, the aim is to solve problems with 1024, 3072 and 4864 variables, and all variables are limited to be $-8 \leq S^{(1)} \leq 8$.

III. MAGA-BigOpt

MAGA was first proposed to deal with large-scale global numerical optimization problems in [5]. In this work, based on the framework of MAGA, we propose MAGA-BigOpt to deal with the big optimization problems. In MAGA-BigOpt, an agent represents a candidate solution $S^{(1)}$ to the given problem. The energy of an agent is equal to the negative value of the objective function F defined in Eq. (5). The purpose of an agent is to increase its energy as much as possible.

All agents live in a latticelike environment L, which is called an agent lattice. The size of L is $L_{size} \times L_{size}$. We label an agent located at (i, j) as $L_{i,j}$, $i, j = 1, 2, \ldots, L_{size}$. It is a vector format of a candidate solution $S^{(1)}$, which is defined in Eq. (6)

$$\begin{aligned} L_{i,j} &= vector(S^{(1)}) \\ &= [S^{(1)}_{11}, S^{(1)}_{12}, \ldots, S^{(1)}_{1M}, S^{(1)}_{21}, S^{(1)}_{22}, \ldots, S^{(1)}_{2M}, \\ &\quad \ldots, S^{(1)}_{N1}, S^{(1)}_{N2}, \ldots, S^{(1)}_{NM}] \end{aligned} \tag{6}$$

For the sake of simplicity, we re-label $L_{i,j}$ as

$$L_{i,j} = [l_1, l_2, \ldots, l_n], \qquad n = N \times M \tag{7}$$

The neighbors of $L_{i,j}$ are defined as

$$\begin{aligned} Neighbors_{i,j} &= \{L_{next(i),j}, L_{i,next(j)}, \\ &\quad L_{prev(i),j}, L_{i,prev(j)}\} \end{aligned} \tag{8}$$

where

$$\begin{aligned} next(x) &= \begin{cases} x+1 & x \neq L_{size} \\ 1 & x = L_{size} \end{cases} \\ prev(x) &= \begin{cases} x-1 & x \neq 1 \\ L_{size} & x = 1 \end{cases} \end{aligned} \tag{9}$$

To maximize the energy, four evolutionary operators are designed for agents. The neighborhood competition operator and the neighborhood crossover operator realize the behaviors of competition and cooperation, respectively. The mutation operator and the self-learning operator realize the behaviors of making use of knowledge.

A. Neighborhood competition operator

Suppose the neighborhood competition operator is conducted on $L_{i,j}$. If $L_{i,j}$ satisfies Eq. (10), it is a winner; otherwise it is a loser.

$$Energy(L_{i,j}) \geq Energy(Max_{i,j}) \tag{10}$$

where $Max_{i,j} = (m_1, m_2, \ldots, m_n)$ is the agent with the highest energy among the neighbors of $L_{i,j}$.

If $L_{i,j}$ is a winner, it can still alive in the agent lattice, which means the value (genes) of $L_{i,j}$ will not change in this operator. If $L_{i,j}$ is a loser, it must be die, and its lattice will be occupied by $Max_{i,j}$. There are two strategies to change the lattice point with the information of $Max_{i,j}$. Selected with probability P_o, strategy 1 is employed if $U(0,1) < P_o$; otherwise strategy 2 will be used, where $U(0,1)$ is a random number uniformly distributed in the range of $(0,1)$.

In the two occupying strategies, a new agent $New_{i,j}$ will be generated by $Max_{i,j}$, and then it will be put in to this lattice.

Strategy 1:

$$\begin{aligned} New_{i,j} &= (e_1, e_2, \ldots, e_n), \\ e_k &= bound_k(m_k + U(-1,1) \times (m_k - l_k)) \end{aligned} \tag{11}$$

where $bound_k(x) = \min(\overline{x_k}, \max(\underline{x_k}, x))$ makes x stay in range $[\overline{x_k}, \underline{x_k}]$. $\overline{x_k}$ and $\underline{x_k}$ correspond to the upper and lower bound value in dimension k. $\min(a, b)$ and $\max(a, b)$ are the minimum and maximum among a and b respectively.

Strategy 2:

$$\begin{aligned} New_{i,j} &= (m_1, m_2, \ldots, m_{i_1-1}, \\ &\quad m_{i_2}, m_{i_2-1}, \ldots, m_{i_1}, \\ &\quad m_{i_2+1}, m_{i_2+2}, \ldots, m_n) \end{aligned} \tag{12}$$

where $1 < i_1 < i_2 < n$.

In this operator, two strategies play different roles. Occupying strategy 1, a kind of heuristic crossover, is in favor of preserving some information of a loser, while occupying strategy 2 is similar to the inversion operator in AEA [8] with the function of random search. Therefore, occupying strategy 1 prefers exploitation while occupying strategy 2 prefers exploration.

B. Neighborhood crossover operator

The neighborhood crossover operator makes use of the information of $L_{i,j}$ and $Max_{i,j}$ straightforwardly in that the searching space is extraordinarily large and the chances of evaluating is not too many. A new agent $New_{i,j} = (e_1, e_2, \ldots, e_n)$ is generated in the way below and takes over $L_{i,j}$ if $L_{i,j}$ does not have the highest energy compared with its neighbor.

$$e_k = \min(l_k, m_k) + U(0,1) \times |l_k - m_k| \tag{13}$$

With a given probability P_c, a number of agents can be checked and replaced.

Holding the propriety that it will not exceed the boundary established by $L_{i,j}$ and $Max_{i,j}$, the new agent can stay in the feasible area naturally. Note that it could only be used in convex feasible region.

C. Mutation operator

A new agent $New_{i,j} = (e_1, e_2, \ldots, e_n)$ is generated from $L_{i,j}$ as follows,

$$e_k = \begin{cases} l_k & U(0,1) < \frac{1}{n} \\ bound_k(l_k + G(0,1/t)) & \text{otherwise} \end{cases} \tag{14}$$

where $G(0,1/t)$ is a Gaussian random number generator with a mean of 0 and a variance of $1/t$ and t is the evolution generation.

This operator is similar to the Gaussian mutation operator used in the evolutionary programming and the evolution strategy, but it only performs a small perturbation on some variables of agent.

D. Self-learning operator

In the self-learning operator, we use the negative sub-gradient as the searching direction. Define $T = A \times S^{(1)}$, and T_q, X_p are the qth row of matrix T and the pth row of matrix X, respectively. The kth element of T_q is

$$T_{qk} = \sum_l A_{ql} S^{(1)}_{lk} \tag{15}$$

The mean and standard deviation of vector T_q is

$$\mu(T_q) = \frac{1}{M} \sum_k T_{qk} \tag{16}$$

$$\sigma(T_q) = \left(\frac{1}{M-1} \sum_k (T_{qk} - \mu(T_q))^2 \right)^{\frac{1}{2}} \tag{17}$$

Thus, the derivative of $\mu(T_q)$ and $\sigma(T_q)$ can be calculated by

$$\frac{\partial \mu(T_q)}{\partial S^{(1)}_{ij}} = \frac{A_{qi}}{M} \tag{18}$$

$$\frac{\partial \sigma(T_q)}{\partial S^{(1)}_{ij}} = \frac{T_{qk} - \mu(T_{qk}) A_{qi}}{(M-1)\sigma(T_q)} \tag{19}$$

C is the correlation matrix, defined in Eq. (2), for each element C_{pq} in C, it is defined as

$$C_{pq} = \frac{covar(X_p, T_q)}{\sigma(X_p) \times \sigma(T_q)} \tag{20}$$

and $covar(X_p, T_q)$ is calculated by Eq.(21)

$$\begin{aligned} &covar(X_p, T_q) = \\ &\tfrac{1}{M} \textstyle\sum_k \left(X_{pk} - \mu(X_p)\right) \left(T_{qk} - \mu(T_q)\right) \end{aligned} \tag{21}$$

then, we can get Eq. (22) as derivate of Eq. (21)

$$\frac{\partial covar(X_p, T_q)}{\partial S^{(1)}_{ij}} = \frac{1}{M} \left(X_{pj} - \mu(X_p)\right) A_{qi} \tag{22}$$

so if $\sigma(T_q) \neq 0$, the derivation of C_{pq} can be calculated as

$$\begin{aligned} \frac{\partial C_{pq}}{\partial S^1_{ij}} &= \frac{\left(X_{pj} - \mu(X_p)\right) A_{qi}}{M \cdot \sigma(X_p)\sigma(T_q)} - \\ &\quad \frac{\left(T_{pj} - \mu(X_p)\right) A_{qi} C_{pq}}{(M-1)\sigma(T_q)^2} \end{aligned} \tag{23}$$

when $\sigma(T_q) = 0$, C_{pq} is not derivable. However, $\sigma(T_q)$ is equal to 0 only when all elements in T_q are the same, and we can simply ignore this particular case and give a sub-gradient of f_1 and f_2.

$$\begin{aligned} \frac{\partial f_1}{\partial S^{(1)}_{ij}} &= \frac{2}{N \times M} \left(S^{(1)}_{ij} - S_{ij} \right) \\ \frac{\partial f_2}{\partial S^{(1)}_{ij}} &= \frac{2}{N^2 - N} \sum_{p,q \neq p} \frac{\partial C_{pq} \cdot C_{pq}}{\partial S^{(1)}_{ij}} + \\ &\quad \frac{2}{N} \sum_p \frac{\partial C_{pp}(C_{pp} - 1)}{\partial S^{(1)}_{ij}} \end{aligned} \tag{24}$$

The sub-gradient of an agent is a matrix G, which is the sum of the derivation of two objective functions.

$$G_{ij} = \frac{\partial f_1}{S^{(1)}_{ij}} + \frac{\partial f_2}{S^{(1)}_{ij}} \tag{25}$$

In the self-learning operator, N_{sl} candidate solution are generated in the direction of the negative sub-gradient with

different distance to $L_{i,j}$. An candidate solution $New_{i,j}^{(k)}$ is calculated by Eq.(26)

$$New_{i,j}^{(k)} = L_{i,j} - \xi_k \frac{\nabla L_{i,j}}{\|\nabla L_{i,j}\|}, \quad k = 1, 2, \ldots, N_{sl} \qquad (26)$$

where $\nabla L_{i,j}$ is the sub-gradient of agent $L_{i,j}$, the vector format of matrix G. $\|\nabla L_{i,j}\|$ is the norm of the sub-gradient. ξ_k is the distance from to $L_{i,j}$. ξ_k is generated by an exponential distribution random number generator, the parameter of the exponential distribution λ is set to be $t/(1000 \cdot \|\nabla L_{i,j}\|)$, t is the evolution generation. The reason of this step selected is that we need higher probability to search in the given direction nearly and can also reach some point far away. All candidates must be kept within the feasible regain, being the same with occupying strategy 1.

After we obtained all the N_{sl} new agents in the negative gradient direction, we choose the one with the largest energy as the result of the self-learning operator and use it to replace $L_{i,j}$.

E. Implementation of MAGA-BigOpt

In MAGA-BigOpt, the neighborhood competition operator is performed on each agent in each generation. As a result, the agents with low energy are cleaned out from the agent lattice so that there is more developing space for the agents with high energy. Then, the neighborhood crossover operator and the mutation operator are performed on each agent with probabilities P_c and P_m, respectively. In order to reduce the computational cost, the self-learning operator is only performed on the best agent in each generation. In general, the four operators utilize different methods to simulate the behaviors of agents, and play different roles in MAGA-BigOpt. Algorithm 1 describes the details of MAGA-BigOpt.

Algorithm 1: MAGA-BigOpt

1 Initialize agent lattice L, update historical best agent $Best$, $t \leftarrow 0$;
2 Perform the neighborhood competition operator on each agent;
3 Perform the neighborhood crossover operator on each agent with probability P_c;
4 Perform the mutation operator on each agent with probability P_m;
5 Find the best agent $CBest$ in the current generation, and perform the self-learning operator on $CBest$;
6 If $Energy(CBest) > Energy(Best)$, $Best \leftarrow CBest$; otherwise $CBest \leftarrow Best$;
7 If termination criteria are reached, output $Best$ and stop; otherwise $t \leftarrow t + 1$, go to step 2.

IV. Experiment

We tested our method on a machine with 2.27 GHz CPU and 4 GB RAM. The operating system is Windows 7 and the development environment is Visual Studio 2010. The parameters of the algorithm in our experiments are learnt from [5] and listed in Table I.

TABLE I
PARAMETER SETTINGS

Parameter	L_{size}	P_o	P_c	P_m	N_{sl}
Value	6	0.2	0.8	0.1	30

The results compared with the baseline algorithm are averaged over 30 independent runs and showed in Tables II and III for the cases without and with noise, respectively. As can be seen, MAGA-BigOpt has an extraordinary performance in these problems both in the quality of obtained solutions and the convergence speed.

TABLE II
THE RESULTS OF MAGA-BIGOPT FOR PROBLEMS WITHOUT NOISE

Problem	Methods	Mean	Var	NFFE	Time(s)
D4	Baseline	1.8709	–	–	–
	MAGA-BigOpt	0.0610	5.50E-7	584.7	0.190
D12	Baseline	2.9294	–	–	–
	MAGA-BigOpt	0.0019	2.10E-6	603.6	1.322
D19	Baseline	3.1854	–	–	–
	MAGA-BigOpt	0.0025	2.43E-6	1346.9	7.176

TABLE III
THE RESULTS OF MAGA-BIGOPT FOR PROBLEMS WITH NOISE

Problem	Methods	Mean	Var	NFFE	Time(s)
D4N	Baseline	1.7414	–	–	–
	MAGA-BigOpt	0.0590	4.28E-7	586.8	0.191
D12N	Baseline	2.8237	–	–	–
	MAGA-BigOpt	0.0018	2.27E-6	645.3	1.396
D19N	Baseline	3.1659	–	–	–
	MAGA-BigOpt	0.0026	2.90E-6	1724.8	9.216

V. Conclusion

In this paper, based on the framework of MAGA, MAGA-BigOpt is designed to deal with big optimization problems proposed in the Optimization of Big Data 2015 Competition. Efficient neighborhood competition, neighborhood crossover and special self-learning operators are designed to realize the behaviors of agents. In the experiments, six benchmark problems from the Optimization of Big Data 2015 Competition are used to validate the performance of MAGA-BigOpt. The experimental results show that MAGA-BigOpt clearly outperforms the base-line algorithm. Moreover, the computational cost of MAGA-BigOpt is really low.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China under Grant 61271301, the Research Fund for the Doctoral Program of Higher Education of China under Grant 20130203110010, and the Fundamental Research Funds for the Central Universities under Grant K5051202052.

REFERENCES

[1] S. K. Goh, H. A. Abbass, K. C. Tan, and A. AlMamun, "Decompositional independent component analysis using multi-objective optimization," *Soft Computing*, pp. 1–16, 2015.

[2] H. A. Abbass, "Calibrating independent component analysis with laplacian reference for real-time eeg artifact removal," in *Neural Information Processing*. Springer, 2014, pp. 68–75.

[3] S. K. Goh, H. A. Abbass, K. C. Tan, and A. Al Mamun, "Artifact removal from eeg using a multi-objective independent component analysis model," in *Neural Information Processing*. Springer, 2014, pp. 570–577.

[4] S. K. Goh, H. A. Abbass, K. C. Tan, and A. A. Mamun, "Evolutionary big optimization (BigOpt) of signals," in *IEEE Congress on Evolutionary Computation, Sendai, Japan*. IEEE, 2015.

[5] W. Zhong, J. Liu, M. Xue, and L. Jiao, "A multiagent genetic algorithm for global numerical optimization," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, no. 2, pp. 1128–1141, April 2004.

[6] J. Liu, W. Zhong, and L. Jiao, "A multiagent evolutionary algorithm for constraint satisfaction problems," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 36, no. 1, pp. 54–73, Feb 2006.

[7] ——, "A multiagent evolutionary algorithm for combinatorial optimization problems," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 40, no. 1, pp. 229–240, Feb 2010.

[8] Z. Pan and L. Kang, "An adaptive evolutionary algorithm for numerical optimization," in *Simulated Evolution and Learning, number 1285 in Lecture Notes in Computer Science*, no. 1285. Springer-Verlag, 1997, pp. 27–34.