Distributed Multiobjective Quantum-inspired Evolutionary Algorithm (DMQEA)

Si-Jung Ryu and Jong-Hwan Kim

Abstract Most of the multiobjective evolutionary algorithm inherently has heavy computational burden, so it takes a long processing time. For this reason, many researches for reducing computational time have been carried out, in particular by using distributed computing such as multi-thread coding, GPU coding, etc. In this paper, multi-thread coding is used to reduce computational time and applied to multiobjective quantum-inspired evolutionary algorithm (MQEA). In MQEA, nondominated sorting and crowding distance assignment which take a long time are carried out in each subpopulation. By multi-thread coding, the processes in each subpopulation can be performed simultaneously. To demonstrate the effectiveness of the proposed distributed MQEA (DMQEA), comparisons with single-thread and multi-thread are carried out for seven DTLZ functions.

Key words: Multiobjective evolutionary algorithm, Distributed computing, Quantum-inspired evolutionray algorithm, multiobjective quantum-inspired evolutionary algorithm

1 Introduction

Quantum-inspired evolutionary algorithm (QEA) employs the probabilistic mechanism inspired by the concept and principles of quantum computing, such as a quantum bit and superposition of states [1, 2, 3]. In addition, multiobjective quantum-inspired evolutionary algorithm (MQEA) was developed with the purpose of solving multiobjective optimization problems [4].

S.-J. Ryu and J.-H. Kim
Department of Electrical Engineering, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon, 305-701, Republic of Korea,
e-mail: {sjryu, johkim}@rit.kaist.ac.kr

MQEA provides high quality solutions close to Pareto-optimal solution set for multiobjective problems. Recently, preference-based sorting was applied to the nondominated solutions in an archive of MQEA to reflect the designer's preference in sorting them and selecting one preferred solution out of them [5, 6]. However, MQEA also has a heavy computational burden like other multiobjective evolutionary algorithms. Especially, MQEA employs the nondominated sorting and crowding distance assiginment in the subpopulations, which leads to heavy computational burden.

There have been many researches for reducing the computational load for the multiobjective optimization problems [7, 8, 9]. This kind of research can be divided into two major issues; algorithmic development and computing power development. Firstly, researches in terms of algorithmic development are progressed by modifying original algorithms efficiently and eliminating unnecessary parts of algorithms. Secondly, researches in terms of computing power development are progressed by distributing the computational burden of algorithms using multi-thread coding or GPU coding. The latter has much outstanding performance compared to the modification of algorithms.

In this paper, distributed computing is applied to develop distributed MQEA (DMQEA) to reduce the computational burden. It is also efficient to apply distributed computing to multiobjective evolutionary algorithms because there are many processes which can be performed simultaneously in the algorithms. Since subpopulation processes of MQEA are independent each other, the processes in each subpopulation can be performed simultaneously. To compare the performance of the proposed DMQEA, experiments are carried out for seven DTLZ functions.

The rest of this paper is organized as follows: MQEA is briefly introduced in Section 2. Section 3 proposes DMQEA. The experimental results are discussed in Section 4 and concluding remarks follow in Section 5.

2 Distributed Computing for MQEA

2.1 QEA

Building block of classical digital computer is represented by two binary states, '0' or '1', which is a finite set of discrete and stable state. In contrast, QEA utilizes a novel representation, called a Q-bit representation [1], for the probabilistic representation that is based on the concept of qubits in quantum computing [10]. Quantum system enables the superposition of such state as follows:

$$\alpha|0\rangle + \beta|1\rangle \tag{1}$$

where α and β are the complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$.

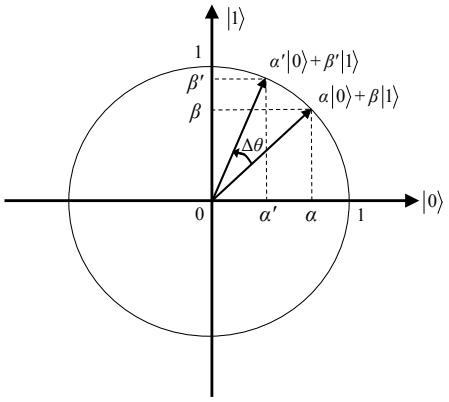

Fig. 1: Qubit described in two dimensional space.

Qubit is shown in Fig. 1, which can be illustrated as a unit vector on the two dimensional space as follows:

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} \tag{2}$$

where $|\alpha|^2 + |\beta|^2 = 1$. Q-bit individual is defined as a string of Q-bits as follows:

$$\mathbf{q}_j^t = \left[\begin{array}{c|c|c|c} \alpha_{j1}^t & \alpha_{j2}^t & \cdots & \alpha_{jm}^t \\ \beta_{j1}^t & \beta_{j2}^t & \cdots & \beta_{jm}^t \end{array}\right] \tag{3}$$

where m is the string length of Q-bit individual, and $j = 1, 2, ..., n$ for the population size n. The population of Q-bit individuals at generation t is represented as $Q(t) = \{\mathbf{q}_1^t, \mathbf{q}_2^t, ..., \mathbf{q}_n^t\}$.

Since Q-bit individual represents the linear superposition of all possible states probabilistically, diverse individuals are generated during the evolutionary process. The procedure of QEA and the overall structure for single-objective optimization problems are described in [1, 2].

2.2 MQEA

Based on QEA, Multiobjective Quantum-inspired Evolutionary Algorithm (MQEA) was developed to solve multiobjective problems [4]. MQEA is designed by incorporating QEA with fast non-dominated sorting and crowding distance assignment. MQEA provides the solutions close to Pareto-optimal solution set for multiobjective problems. Overall procedure of MQEA is summarized in Algorithm 1. Each step is described in the following.

Algorithm 1 Procedure of MQEA

```
1: t ⟵ 0
2: Initialize Q_k(t)
3: Observe the states of Q_k(t) and form P_k(t)
4: Evaluate P_k(t) and store all solutions in P_k(t) into P(t)
5: Copy the nondominated solutions in P(t) to A(t)
6: while (not termination condition) do
7:    t ⟵ t + 1
8:    Make P_k(t) by observing the states of Q_k(t − 1)
9:    Evaluate P_k(t)
10:   Form P_k(t) through the fast nondominated sorting and crowding distance sorting
11:   Store all solutions in every P_k(t) into P(t)
12:   Form A(t) by nondominated solutions in A(t − 1) ∪ P(t)
13:   Migrate randomly selected solutions in A(t) to every R_k(t)
14:   Update Q_k(t) using Q-gates referring to the solutions in R_k(t)
15: end while
```

1), 2) In this step, $Q_k(0)$ is initialized with $1/\sqrt{2}$, where $i = 1, 2, ..., m$, $j = 1, 2, ..., n$, and $k = 1, 2, ..., s$. Note that m is the string length of Q-bit individual, n is the subpopulation size, and s is the number of subpopulations.

3) Binary solutions in $P_k(0)$ are formed by observing the states of $Q_k(0)$. One binary solution has a value either 0 or 1 according to the probability either $|\alpha_i^0|$ or $|\beta_i^0|$ as follows:

$$x_i^0 = \begin{cases} 0 \text{ if rand}[0,1] \leq |\alpha_i^0|^2 \\ 1 \text{ if rand}[0,1] > |\alpha_i^0|^2. \end{cases} \tag{4}$$

4) Each binary solution, $\mathbf{x_j^0}$, in $P_k(0)$ is evaluated. All the solutions in $P_k(0)$ are stored in $P(0)$.

5) Archive $A(0)$ is filled with nondominated solutions in $P(0)$.

6), 7) The process terminates if the number of generation reaches the termination number.

8), 9) Binary solutions in $P_k(t)$ are generated through the multiple observing the states of $Q_k(t-1)$ and fitness values are calculated for each binary solution.

10) Individuals in the previous population and current population are sorted by the fast nondominated sorting and the crowding distance sorting and select n individuals [11]. $P_k(t)$ is formed with n selected individuals.

11) All solutions in every $P_k(t)$ are copied to $P(t)$.

12) An archive $A(t)$ is formed by nondominated solutions in the previous archive and global population $(A(t-1) \cup P(t))$

13) Solutions in current archive are randomly selected and solutions in every reference population are randomly replaced by the selected solutions. Global random migration procedure occurs at each and every generation.

14) Fitness values in each subpopulation are compared, and then decided the update direction of Q-bit individuals. the rotation gate $U(\Delta\theta)$ is employed as an update operator for Q-bit individuals, which is defined as follows:

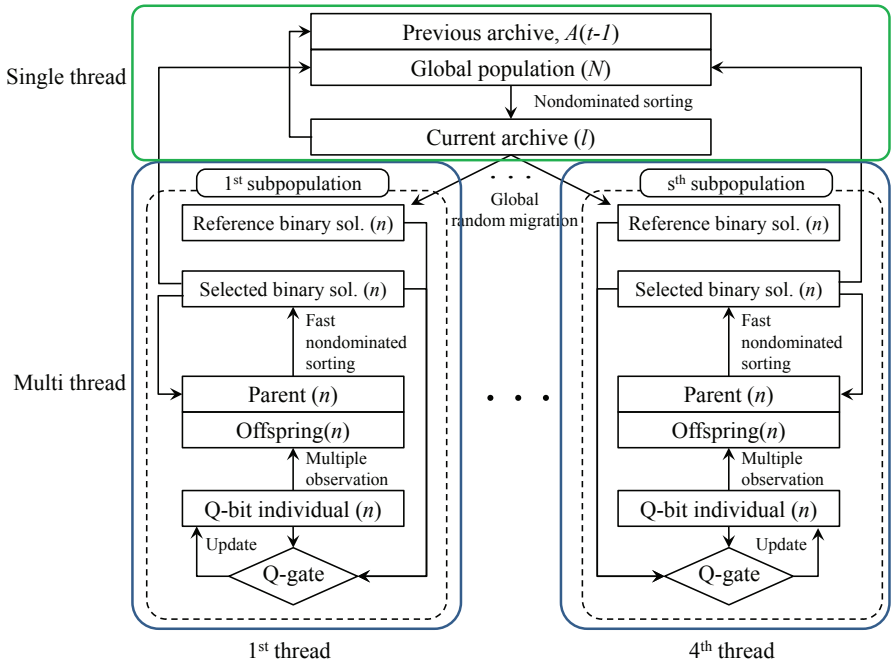

Fig. 2: Procedure of the DMQEA.

$$\mathbf{q_j^t} = \mathbf{U}(\boldsymbol{\Delta}\theta) \cdot \mathbf{q_j^{t-1}} \tag{5}$$

with

$$U(\Delta\theta) = \begin{bmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) \\ \sin(\Delta\theta) & \cos(\Delta\theta) \end{bmatrix}$$

where $\Delta\theta$ is the rotation angle of each Q-bit.

3 DMQEA

In recent years, computer-related technological advances such as multi-threading and GPU processing enable to reduce the computing time significantly. In this paper, DMQEA is proposed by applying multi-thread coding into MQEA. The main difference of DMQEA compared to MQEA is that each and every subpopulation, depicted in Fig. 2, goes through the evolutionary process simultaneously. The details of the subpopulation process of the DMQEA are as follows. Each subpopulation process is performed by the corresponding thread. In this paper, four subpopulations are executed at one time because experimental PC provides four threads. In this regard, thread

Table 1: Parameter setting of the DMQEA for DTLZ problems

Parameters	Values
The number of generations	3,000
The number of subpopulations (s)	4
The number of multiple observations	10
The rotation angle ($\Delta\theta$)	0.23π

synchronization is needed in each and every generation because computational times of threads are all different. Therefore, it is required to wait until all the thread processes are finished. And then, obtained solutions from the subpopulations are stored in the global population. The rest processes of DMQEA such as archive generation and migration are executed in single-thread because they are affected by the other processes.

4 Experimental Results

Experiments were carried out under Intel Core i5 650 CPU which provides four threads. Clock speed of PC is 3.20GHz and operating system is Windows 7 32bit. Parameter setting for experiments is given in Table 1. The experiments were carried out under three conditions of different sizes of the subpopulation and different numbers of objectives. The number of variables for each DTLZ function was set to 11 for DTLZ1, 16 for DTLZ2 to DTLZ6, and 26 for DTLZ7 function if the number of objectives is seven. Otherwise, the number of variables for each DTLZ function was set to 7 for DTLZ1, 12 for DTLZ2 to DTLZ6, and 22 for DTLZ7 function.

Table 2 shows the computation time of processing MQEAs by single-thread and multi-thread with three different conditions on subpopulation size and the number of objectives. As shown in the table, the processing time computed by multi-thread is about 70% lower than that by single-thread.

5 Conclusion

This paper proposed DMQEA to reduce the computational time of MQEA. The main difference of DMQEA compared to MQEA was that the processes in each subpopulation are executed in the multiple threads. The reason why distributed computing was applied to subpopulation processes is that they contains heavy computational processes such as nondominated sorting and crowding distance assignment. To demonstrate the effectiveness of the proposed DMQEA, comparisons of MQEAs with single-thread and multi-thread

Table 2: Comparisons of computation times between single and multi-thread MQEAs for seven DTLZ functions (unit: second)

(a) subpopulation size: 25, objectives: 7

Problem	Single-thread	Multi-thread
DTLZ1	501.0	315.1
DTLZ2	532.0	370.6
DTLZ3	585.1	446.0
DTLZ4	515.6	308.6
DTLZ5	528.4	414.6
DTLZ6	539.9	442.8
DTLZ7	591.1	473.4
Average	541.9	395.9

(b) subpopulation size: 25, objectives: 3

Problem	Single-thread	Multi-thread
DTLZ1	427.1	298.1
DTLZ2	455.3	304.3
DTLZ3	492.8	361.1
DTLZ4	445.9	245.9
DTLZ5	479.5	324.3
DTLZ6	471.5	349.1
DTLZ7	498.5	354.3
Average	467.2	319.6

(c) subpopulation size: 50, objectives: 3

Problem	Single-thread	Multi-thread
DTLZ1	1438.8	1013.5
DTLZ2	1466.6	1017.7
DTLZ3	1513.3	1331.5
DTLZ4	1452.0	987.5
DTLZ5	1515.3	1259.4
DTLZ6	1498.9	969.7
DTLZ7	1596.4	1032.5
Average	1497.3	1087.4

were carried out for seven DTLZ functions. Results showed that DMQEA reduces an execution time significantly. However, DMQEA still has a problem of computation time to be applied to real time applications. Therefore, it needs to be more improved as a future work.

Acknowledgments.

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the Human Resources Development Program for Convergence Robot Specialists support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2012-H1502-12-1002).

References

1. Han, K.-H. and Kim, J.-H. (2002) Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE Trans Evol Computat 6(6): 580–593.
2. Han, K.-H. and Kim, J.-H. (2004) Quantum-inspired evolutionary algorithms with a new termination criterion, Hϵ gate, and two phase scheme. IEEE Trans Evol Computat 8(2): 156–169.
3. Han, K.-H. and Kim, J.-H. (2006) On the analysis of the quantum-inspired evolutionary algorithm with a single individual. Paper presented at IEEE Congress Evolutionary Computation, pp. 9172–9179, 2006.
4. Kim, Y.-H., Kim, J.-H. and Han, K.-H. (2006) Quantum-inspired multiobjective evolutionary algorithm for multiobjective 0/1 knapsack problems. Paper presented at IEEE Congress Evolutionary Computation, pp. 2601–2606, 2006.
5. Kim, J.-H., Han, J.-H., Kim, Y.-H., Choi, S.-H. and Kim, E.-S. (2012) Preference-based Solution Selection Algorithm for Evolutionary Multiobjective Optimization. IEEE Trans Evol Computat 16(1): 20-34.
6. Ryu, S.-J., Lee, K.-B. and Kim, J.-H. (2012) Improved version of a multiobjective quantum-inspired evolutionary algorithm with preference-based selection. Paper presented at IEEE Congress Evolutionary Computation, pp. 1–7, 2012.
7. Tan, K.C., Yang, Y.J. and Goh, C.K. (2006) A distributed Cooperative coevolutionary algorithm for multiobjective optimization. IEEE Trans Evol Computat 10(5): 527–549.
8. Deb, K., Zope, P. and Jain, A. (2003) Distributed Computing of Pareto-Optimal Solutions with Evolutionary Algorithms. Evolutionary Multi-Criterion Optimization, LNCS 2632: 534–549.
9. Tan, K.C., Tay, A. and Cai, J. (2003) Design and implementation of a distributed evolutionary computing software. IEEE Trans Syst Man Cybern. C, Appl 33(3): 325-338.
10. Hey, T. (1999) Quantum computing: an introduction. Computing and Control Eng J 10(3): 105–112.
11. Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Computat 6(2): 182–197.
12. Zitzler, E. (1999) Evolutionary algorithms for multiobjective optimization: methods and applications. Berichte aus der Informatik, Shaker Verlag, Aachen-Maastricht.