Asynchronous differential evolution with self-adaptive parameter control for global numerical optimization

Tae Jong Choi¹, and Yeonju Lee^{2,*}

¹Department of Computer Engineering, Sungkyunkwan University, Republic of Korea

²CTO Department, LG Electronics, Republic of Korea

Abstract. In this paper, we propose an extended self-adaptive differential evolution algorithm, called A-jDE. A-jDE algorithm is based on jDE algorithm with the asynchronous method. jDE algorithm is one of the popular DE variants, which shows robust optimization performance on various problems. However, jDE algorithm uses a slow mutation strategy so that its convergence speed is slow compared to several state-of-the-art DE algorithms. The asynchronous method is one of the recently investigated approaches that if it finds a better solution, the solution is included in the current population immediately so it can be served as a donor individual. Therefore, it can improve the convergence speed significantly. We evaluated the optimization performance of A-jDE algorithm in 13 scalable benchmark problems on 30 and 100 dimensions. Our experiments prove that incorporating jDE algorithm with the asynchronous method can improve the optimization performance in not only a unimodal benchmark problem but also multimodal benchmark problem significantly.

1 Introduction

Differential Evolution (DE) algorithm proposed by Storn and Price [1] is a popular evolutionary algorithm especially for solving continuous domain optimization problems. DE algorithm is popular among many evolutionary algorithms thanks to following two features. First, the structure of DE algorithm is simple and easy to implement so researchers and practitioners can apply DE algorithm to solve their problems easily. Second, DE algorithm usually finds more accurate solutions than other population-based metaheuristics, which has been demonstrated on many optimization benchmark problems, i.e., IEEE CEC, as well as real-world problems such as neural network learning [2, 3], generating artworks [4]. Since DE algorithm was proposed, many studies have been conducted for improving its searchability, such as adaptive parameter control [5-10], adaptive strategy control [11], and hybridizing DE algorithm with other methods [12], to solve more complicated optimization problems. Zhabitskaya and Zhabitsky proposed a modified DE algorithm called Asynchronous DE (ADE) [12], which is designed for parallel optimization. In contrast to

* Corresponding author: yeonju.lee@lge.com

classical DE algorithm that generates trial vectors of all the target vectors of the current population at each generation, ADE algorithm generates a trial vector with a randomly selected target vector or the worst individual with respect to the fitness value of a given optimization problem. Although ADE algorithm was designed for the parallel optimization, its optimization performance shows competitive to classical DE algorithm even in the sequential mode. There are several interesting questions remain. For example, it is an interesting question whether the asynchronous method still is useful when it is applied to the state-of-the-art DE algorithms. In this paper, we extend jDE algorithm [5], which is one of the state-of-the-art DE algorithms, with the asynchronous method and test its optimization performance in 13 scalable benchmark problems [13] on 30 and 100 dimensions. Although there exist many other automatically tuning control parameter methods, we selected jDE algorithm because its self-adaptive parameter control method is suitable for the parallel optimization. The optimization performance of the proposed algorithm outperformed the compared algorithms significantly. This implies that the self-adaptive parameter control method still works well along with asynchronous DE algorithm.

2 Related Work

2.1 Classical DE Algorithm

Since DE algorithm is one of the evolutionary algorithms, it has the same operators, mutation, crossover, and selection. In DE algorithm, the mutation and crossover operators generate a new candidate solution from a current solution, and the selection operator chooses the better solution between the current and candidate solutions as a member of next generation. Therefore, the mutation and crossover operators increase the diversity of solutions, and the selection operator decreases it. DE algorithm has NP individuals, and each individual is a D -dimensional vector. At the beginning of its evolutionary process, DE algorithm scatters each individual in the search space of a given problem uniformly. The i th individual is denoted by $X_i^G = \{x_{i,1}^G, x_{i,2}^G, \dots, x_{i,D}^G\}$ and the lower boundary and the upper boundary of the search space are denoted by $X_{min} = \{x_{min,1}, x_{min,2}, \dots, x_{min,D}\}$ and $X_{max} = \{x_{max,1}, x_{max,2}, \dots, x_{max,D}\}$, respectively. Each individual is initialized as follows: $x_{i,j}^0 = x_{min,j} + rand_{i,j} \cdot (x_{max,j} - x_{min,j})$ where $rand_{i,j}$ represents a uniform random number within the range $[0,1]$. After that DE algorithm repeats three evolutionary operators, the mutation, crossover, and selection, until one of the termination criteria is met. In this section, we describe one of the most commonly used mutation and crossover operators called DE/rand/1/bin. In DE/rand/1/bin, a mutant individual $V_i^G = \{v_{i,1}^G, v_{i,2}^G, \dots, v_{i,D}^G\}$ is generated by three donor individuals, which is calculated as follows:

$$v_{i,j}^G = x_{r_1}^G + F \cdot (x_{r_2}^G - x_{r_3}^G) \quad (1)$$

where F represents the scaling factor and $x_{r_1}^G$, $x_{r_2}^G$, and $x_{r_3}^G$ represent the three donor individuals, and $i \neq r_1 \neq r_2 \neq r_3$. After the mutation operator, a trial individual $U_i^G = \{u_{i,1}^G, u_{i,2}^G, \dots, u_{i,D}^G\}$ is generated by the target and mutant individuals as follows:

$$u_{i,j}^G = \begin{cases} v_{i,j}^G, & rand_{i,j} < CR \text{ or } j == j_{rand} \\ u_{i,j}^G, & \text{otherwise} \end{cases} \quad (2)$$

where CR represents the crossover rate and j_{rand} represents a uniform random index within the range $[1,D]$, which makes it possible for the trial individual to have at least one of the mutant elements. After the crossover operator, the selection operator chooses the member

of next generation between the target and trial individuals with respect to their fitness values as follows:

$$X_i^{G+1} = \begin{cases} U_i^G, & f(U_i^G) \leq f(X_i^G) \\ X_i^G, & \text{otherwise} \end{cases} \quad (3)$$

2.2 Asynchronous DE Algorithm

Classical DE algorithm has the concept of generation. In other words, classical DE algorithm generates trial individuals of all of the target individuals and compares the trial individual with its corresponding target individual. In contrast, ADE algorithm does not have the concept of generation. So, ADE algorithm generates a trial individual of a target individual and becomes a member of the current population immediately if the trial individual has a better fitness value than the target individual has. In terms of selecting a target individual, ADE algorithm has two options. First, ADE algorithm can choose a target individual randomly from the current population. Second, ADE algorithm can select the worst individual of the current population as a target vector. ADE algorithm has two advantages over classical DE algorithm. First, classical DE algorithm runs synchronous-based approach, so it cannot apply a better solution immediately. On the other hand, if ADE algorithm finds a better solution, the solution is included in the current population immediately so it can be served as a donor individual. Second, because ADE algorithm does not require a synchronous population, it is easier to be applied in the parallel optimization.

3 Asynchronous DE with Self-Adaptive Parameter Control

The optimization performance of DE algorithm is significantly affected by the control parameters. Applying poor control parameters to DE algorithm may get stuck in a local optimum or make its convergence speed significantly slow. Therefore, it is important to apply proper control parameters to DE algorithm to achieve a satisfiable optimum. However, finding proper control parameters requires much computational cost because the proper control parameters vary on a given problem. In order to solve this problem, many adaptive and self-adaptive parameter controls that are automatically tuning control parameters have been studied. jDE algorithm is one of the well-known self-adaptive parameter control, which adjusts two control parameters, scaling factor and crossover rate. In jDE algorithm, each individual has its own control parameters F_i and CR_i , and if an individual has better control parameters, then the individual is more likely to has better fitness value. And these individuals should propagate their control parameters to the next generation. The control parameters of each individual are calculated as follows:

$$F_i^{G+1} = \begin{cases} F_i + rand_1 \cdot F_w, & rand_2 \leq \tau_1 \\ F_i^G, & \text{otherwise} \end{cases} \quad (4)$$

$$CR_i^{G+1} = \begin{cases} rand_3, & rand_4 \leq \tau_2 \\ CR_i^G, & \text{otherwise} \end{cases} \quad (5)$$

where $rand_k$ for $k \in \{1,2,3,4\}$ denotes uniform random numbers within the range $[0,1]$. In this paper, we extend jDE algorithm with the asynchronous method. jDE algorithm uses a robust but slow mutation strategy, DE/rand/1. Although this mutation strategy helps jDE algorithm to prevent it from getting stuck in a local optimum, its convergence speed significantly slow. Therefore, by incorporating jDE algorithm with the asynchronous method, its convergence speed can be improved.

4 Simulation

To evaluate the optimization performance of the extended algorithm, we compared it with jDE algorithm without the asynchronous method. We conducted our experiments in the scalable benchmark problems in [13]. Many single objective optimization algorithms were tested in the benchmark problems to measure their optimization performances. We conducted a total of 100 independent experiments in the benchmark problems on 30 and 100 dimensions. In addition, we used population size as 100 for 30-dimensional problems and 400 for 100-dimensional problems. Table 1 presents the experimental results on 30 dimensions. In this table, the better results are marked as boldface. As we can see from this table, jDE algorithm with the asynchronous method outperformed original jDE algorithm. In detail, jDE algorithm with the asynchronous method outperformed all of the unimodal benchmark problems ($F_1 \sim F_4$) significantly. This result is natural because the asynchronous method applies a better solution immediately that increases the convergence speed significantly. Also, the unimodal benchmark problems have no local optimum so increasing the convergence speed is safe from getting stuck in a local optimum. Regarding the multimodal benchmark problems, still, jDE algorithm with the asynchronous method outperformed original jDE algorithm in several benchmark problems (F_7 , F_{12} , and F_{13}). In the rest benchmark problems, both of two algorithms get the same optimization results. This result implies that the asynchronous method still works in the multimodal benchmark problems without degrading the optimization performance of jDE algorithm. In addition, table 2 presents the experimental results on 100 dimensions. Again, jDE algorithm with the asynchronous method outperformed original jDE algorithm. In detail, jDE algorithm with the asynchronous method outperformed all of the unimodal benchmark problems ($F_1 \sim F_4$) significantly. Regarding the multimodal benchmark problems, still, jDE algorithm with the asynchronous method outperformed original jDE algorithm in several benchmark problems (F_5 , F_7 , F_{12} , and F_{13}). In the rest benchmark problems, both of two algorithms get the same optimization results. The interesting point of this result is that jDE algorithm with the asynchronous method finds a better solution than original jDE algorithm on Generalized Rosenbrock's Function (F_5). This result implies that the asynchronous method works better in the multimodal benchmark problems even though it increases the greediness of jDE algorithm.

Table 1. The Experimental on 30 Dimensions.

F	MAX GEN	A-jDE		jDE	
		MEAN	STD	MEAN	STD
F_1	1500	1.47E-31	2.12E-31	2.02E-29	2.06E-29
F_2	2000	5.48E-26	4.07E-26	1.54E-24	1.00E-24
F_3	5000	2.08E-15	5.71E-15	1.60E-14	3.45E-14
F_4	5000	2.81E-16	1.44E-15	1.47E-14	9.99E-14
F_5	20000	1.60E-01	7.86E-01	1.60E-01	7.86E-01
F_6	1500	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_7	3000	3.21E-03	8.10E-04	3.41E-03	8.70E-04
F_8	9000	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_9	5000	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_{10}	1500	3.22E-15	6.09E-16	3.29E-15	7.78E-16
F_{11}	2000	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_{12}	1500	2.16E-32	1.76E-32	9.08E-31	1.73E-30

<i>F</i>	MAX GEN	A-jDE		jDE	
		MEAN	STD	MEAN	STD
<i>F</i> ₁₃	1500	1.03E-31	1.26E-31	9.20E-30	9.74E-30

Table 2. The Experimental on 100 Dimensions.

<i>F</i>	MAX GEN	A-jDE		jDE	
		MEAN	STD	MEAN	STD
<i>F</i> ₁	1500	7.60E-17	2.87E-17	1.21E-15	4.61E-16
<i>F</i> ₂	2000	8.74E-17	2.49E-17	7.82E-16	2.20E-16
<i>F</i> ₃	5000	2.24E-02	1.24E-02	4.01E-02	2.45E-02
<i>F</i> ₄	5000	5.39E-10	1.15E-10	8.11E-10	1.52E-10
<i>F</i> ₅	20000	6.68E-06	3.33E-05	1.26E-02	8.22E-02
<i>F</i> ₆	1500	0.00E+00	0.00E+00	0.00E+00	0.00E+00
<i>F</i> ₇	3000	7.98E-03	9.86E-04	8.08E-03	9.32E-04
<i>F</i> ₈	9000	1.09E-10	5.20E-26	1.09E-10	5.20E-26
<i>F</i> ₉	5000	0.00E+00	0.00E+00	0.00E+00	0.00E+00
<i>F</i> ₁₀	1500	1.76E-14	2.04E-15	4.31E-14	6.55E-15
<i>F</i> ₁₁	2000	0.00E+00	0.00E+00	0.00E+00	0.00E+00
<i>F</i> ₁₂	1500	2.29E-28	1.34E-28	1.34E-26	6.59E-27
<i>F</i> ₁₃	1500	2.49E-27	1.25E-27	1.70E-25	8.15E-26

5 Conclusion

In this paper, we presented an extended self-adaptive DE algorithm, A-jDE. jDE algorithm shows robust optimization performance on various problems, but it uses the slow mutation strategy, DE/rand/1. Therefore, we applied jDE algorithm the asynchronous method to increase the greediness of its searchability. The asynchronous method is one of the recently investigated approaches. In contrast to original DE algorithm, the asynchronous method generates a trial individual of a target individual and becomes a member of the current population immediately if the trial individual has a better fitness value than the target individual has. By doing this, if the asynchronous method finds a better solution, the solution is included in the current population immediately so it can be served as a donor individual. Therefore, it can improve the convergence speed significantly. We evaluated the optimization performance of the extended DE algorithm in 13 scalable benchmark problems on 30 and 100 dimensions. Our experiments prove that incorporating jDE algorithm with the asynchronous method can improve the optimization performance in not only a unimodal benchmark problem but also multimodal benchmark problem significantly. Note that, in the experimental results, there is no degrading optimization performance with incorporating jDE algorithm with the asynchronous method. Therefore, it is highly recommended to use A-jDE algorithm instead of original jDE algorithm. In the future study, we will investigate theoretical analysis of the asynchronous method. Theoretical analysis is critical to understand the characteristics of the asynchronous method. In addition, we will extend A-jDE algorithm for solving more complex problems such as constraint optimization, multiobjective optimization, and/or large-scale optimization.

Acknowledgement

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2017R1C1B2012752).

References

1. Storn, Rainer, and Kenneth Price. "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces." *Journal of global optimization* 11.4 (1997): 341-359.
2. Choi, Tae Jong, and Chang Wook Ahn. "An Improved Differential Evolution Algorithm and Its Application to Large-Scale Artificial Neural Networks." *Journal of Physics: Conference Series*. Vol. 806. No. 1. IOP Publishing, 2017.
3. Choi, Tae Jong, and Chang Wook Ahn. "Adaptive Cauchy Differential Evolution with Strategy Adaptation and Its Application to Training Large-Scale Artificial Neural Networks." *International Conference on Bio-Inspired Computing: Theories and Applications*. Springer, Singapore, 2017.
4. Choi, Tae Jong, and Chang Wook Ahn. "Artificial life based on boids model and evolutionary chaotic neural networks for creating artworks." *Swarm and Evolutionary Computation* (2017).
5. Brest, Janez, et al. "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems." *IEEE transactions on evolutionary computation* 10.6 (2006): 646-657.
6. Choi, Tae Jong, Chang Wook Ahn, and Jinung An. "An adaptive cauchy differential evolution algorithm for global numerical optimization." *The Scientific World Journal* 2013 (2013).
7. Choi, Tae Jong, and Chang Wook Ahn. "An adaptive differential evolution algorithm with automatic population resizing for global numerical optimization." *Bio-Inspired Computing-Theories and Applications*. Springer, Berlin, Heidelberg, 2014. 68-72.
8. Choi, Tae Jong, and Chang Wook Ahn. "An adaptive population resizing scheme for differential evolution in numerical optimization." *Journal of Computational and Theoretical Nanoscience* 12.7 (2015): 1336-1350.
9. Choi, Tae Jong, and Chang Wook Ahn. "An adaptive cauchy differential evolution algorithm with population size reduction and modified multiple mutation strategies." *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems-Volume 2*. Springer, Cham, 2015.
10. Choi, Tae Jong, and Chang Wook Ahn. "Adaptive α -stable differential evolution in numerical optimization." *Natural Computing* 16.4 (2017): 637-657.
11. Choi, Tae Jong, and Chang Wook Ahn. "An Adaptive Cauchy Differential Evolution Algorithm with Bias Strategy Adaptation Mechanism for Global Numerical Optimization." *JCP* 9.9 (2014): 2139-2145.
12. Zhabitskaya, Evgeniya, and Mikhail Zhabitsky. "Asynchronous differential evolution." *Mathematical Modeling and Computational Science*. Springer, Berlin, Heidelberg, 2012. 328-333.
13. Yao, Xin, Yong Liu, and Guangming Lin. "Evolutionary programming made faster." *IEEE Transactions on Evolutionary computation* 3.2 (1999): 82-102.