Applied Soft Computing 13 (2013) 2947–2959

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

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

Distributed memetic differential evolution with the synergy of Lamarckian and Baldwinian learning

Chunmei Zhang[a,b], Jie Chen[a,c], Bin Xin[a,d,*]

[a] School of Automation, Beijing Institute of Technology, Beijing 100081, China
[b] School of Electronic Information Engineering, Taiyuan University of Science and Technology, Taiyuan 030024, China
[c] Key Laboratory of Complex System Intelligent Control and Decision, Ministry of Education, Beijing 100081, China
[d] Decision and Cognitive Sciences Research Centre, Manchester Business School, The University of Manchester, Manchester M15 6PB, UK

ARTICLE INFO

Article history:
Received 8 October 2011
Received in revised form 4 February 2012
Accepted 28 February 2012
Available online 16 March 2012

Keywords:
Distributed differential evolution
Memetic algorithm
Lamarckian learning
Baldwinian learning
Hooke–Jeeves algorithm

ABSTRACT

As a population-based optimizer, the differential evolution (DE) algorithm has a very good reputation for its competence in global search and numerical robustness. In view of the fact that each member of the population is evaluated individually, DE can be easily parallelized in a distributed way. This paper proposes a novel distributed memetic differential evolution algorithm which integrates Lamarckian learning and Baldwinian learning. In the proposed algorithm, the whole population is divided into several subpopulations according to the von Neumann topology. In order to achieve a better tradeoff between exploration and exploitation, the differential evolution as an evolutionary frame is assisted by the Hooke–Jeeves algorithm which has powerful local search ability. We incorporate the Lamarckian learning and Baldwinian learning by analyzing their characteristics in the process of migration among subpopulations as well as in the hybridization of DE and Hooke–Jeeves local search. The proposed algorithm was run on a set of classic benchmark functions and compared with several state-of-the-art distributed DE schemes. Numerical results show that the proposed algorithm has excellent performance in terms of solution quality and convergence speed for all test problems given in this study.

1. Introduction

The differential evolution (DE) is a stochastic, population-based global search and optimization method [1]. It uses the difference of solutions to create new candidate solutions and one-to-one spawning competition scheme to select new individuals greedily. These attractive characteristics make DE retain the knowledge of good solutions in the current population. DE, just like the particle swarm optimization (PSO) and genetic algorithm (GA) which have been implemented to various domains [2,3], has been proven to have a good performance on many real-world problems [4]. DE is good at exploring the search space and locating the region of global optima, but it is slow at fine-tuning the solution [5]. Some modifications on the basic DE can improve its performance, which can be grouped into two categories. One depends on the modifications of DE itself including its parameters, operator, and population structure [6,7]; and the other focuses on hybridizing DE with different optimizers such as additional local searchers and other population-based metaheuristics (e.g. particle swarm optimizer) [8–10].

In the case of modifying the population structure of DE, a popular way is dividing the whole population into multiple subpopulations which evolve independently and exchange information mutually. An important motivation behind the multiple-population strategy is to maintain population diversity and achieve the parallel search of the solution space. DE can be easily parallelized due to the fact that each member of the population is evaluated individually. Tasoulis et al. [11] explored how differential evolution can be parallelized by using a unidirectional ring topology and proposed an algorithm, namely parallel differential evolution (PDE), to improve both the speed and the performance of the method. Besides, a lot of research investigated the migration policy of parallel differential evolution, including migration schemes, migration frequencies, the number and size of subpopulations, and so on. Specially, for solving the learning issue in fuzzy neural inference system, Singh et al. [12] studied the parameterization of a parallel distributed DE in detail and discussed the influence of different interval and sizes of migration, as well as different number of islands. Apolloni et al. [13] designed a modified version of the PDE in a generic way, namely island based distributed differential evolution (IBDDE). A set of five parameters was integrated to elaborate on the main principles of the migration. A distributed

* Corresponding author at: School of Automation, Beijing Institute of Technology, Beijing 100081, China. Tel.: +86 1068912463; fax: +861068918233.
E-mail addresses: zcm10606@163.com (C. Zhang), chenjie@bit.edu.cn (J. Chen), brucebin@bit.edu.cn (B. Xin).

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2012.02.028

version of the differential evolution (DDE) was coupled with affine transformation and mutual information maximization to perform the registration of remotely sensed images in [14]. This algorithm differs from PDE and IBDDE by the topology it adopts. Instead of a unidirectional ring, DDE uses a locally connected topology named torus topology. Based on comparative experiments, Falco et al. indicated that DDE is very promising to achieve better performance. Additionally, Izzo et al. [15] presented a heterogeneous asynchronous island model for DE. The results confirmed that such a model could improve the reliability and speed of the algorithm and find significantly better solutions. Recently, Matthieu et al. [16] designed an adaptive mechanism for the scale factor in distributed differential evolution schemes, called "F" adaptive control parallel differential evolution (FACPDE). The empirical results in [14] showed that the employment of multiple scale factors can greatly improve the performance of the distributed algorithm.

In the context of search and optimization, it is worth noting that the key design issue of evolutionary algorithms lies in the successful promotion of tradeoff between exploration and exploitation [17]. Krasnogor and Smith [18] pointed out that the population-based intelligent methods combining local search methods, called memetic algorithms (MAs) which were originally proposed by Moscato and Norman for the traveling salesman problem [19], can lead to a better tradeoff between exploration and exploitation and thereby improved performance. Now, the term MA is widely employed as a synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem solving [20]. There are two mechanisms on how learning influences evolution. One is the Lamarckian learning (L-learning) in which the characteristics of phenotype and genotype acquired by an organism during its lifetime is transferred and can be passed on to the organism's offspring directly. Another is the Baldwinian learning (B-learning), different from the L-learning that the acquired genotype traits are not inherited to its offspring directly, but adaptive learning can guide the course of evolution indirectly in a way that learning alters the shape of search space and thereby provides good evolutionary paths towards individuals. The experimental results of [20] show that the L-learning in general has a higher performance than the B-learning, and most MAs employ the L-learning mechanism to achieve the combination of global search and local search. At the same time, Nguyen et al. [20] also pointed out that the L-learning cannot distinguish individuals effectively, easily leading to stagnation. Though comparatively time-consuming, it is easier to obtain global optima by the B-learning.

In view of above, this paper proposes a novel distributed memetic differential evolution (abbr., DMDE) which integrates the L-learning and the B-learning. In the proposed algorithm, the initial population is distributed over multiple subpopulations according to the von Neumann topology. All subpopulations interact by migrating their respective best individual to neighboring subpopulations and replacing the worst ones partly or entirely. In the evolutionary loop, DE is in charge of global search and the Hook–Jeeves algorithm assists DE to achieve local improvement. In order to balance exploration and exploitation, we incorporate the L-learning and the B-learning by analyzing their characteristics of individual learning. Then we achieve the cooperation in two aspects: one is the migration among subpopulations and the other is the hybridization of DE and the Hook–Jeeves algorithm.

The presented algorithm was run on a set of benchmark problems and compared with several distributed DE schemes. Numerical results show that the proposed DMDE makes a good tradeoff between exploration and exploitation and performs better than three state-of-the-art distributed DE algorithms for tackling both unimodal and multimodal problems.

The remainder of this paper is organized as follows. In Section 2, a basic DE algorithm is briefly introduced. Section 3 proposes the new distributed DE—DMDE. Section 4 analyses the computational complexity of the proposed scheme and presents the experimental results on benchmark functions. This section gives computational complexity and performance comparisons with several state-of-the-art distributed DE schemes as well as a discussion of the obtained results. Conclusion is summarized in Section 5.

2. Basic differential evolution

Differential evolution (DE) is a population-based metaheuristic. DE/rand/1/bin is one of the classic and most successful DE variants [21]. It generates new solution vectors by adding the weighted difference of two randomly selected population members to the third member, and forms the final trial vector with binomial crossover. In addition, DE employs a one-to-one spawning logic which allows replacement of an individual only if the offspring has better fitness value than its corresponding parent.

DE evolves NP D-dimensional individual vectors $\mathbf{x}_{i,g}$, $i = 1, 2, \ldots, NP$, where g denotes the current generation, and NP is the population size. The initial population is randomly generated within the whole search space. After initialization, DE performs in sequence three vector operations: differential mutation, crossover and selection.

A number of variations to the classic DE have been developed. Different DE strategies differ in the way that the base vector is selected, the number of difference vectors used, and the way that crossover points are determined. In order to characterize these variations, a general notation is adopted, namely DE/x/y/z [21]. x represents a string denoting the vector to be perturbed, y is the number of difference vectors considered for perturbation of x, and z is the type of crossover being used. A brief introduction on the well-known DE variant DE/rand/1/bin is given in the following.

2.1. Differential mutation

For each target vector $\mathbf{x}_{i,g}$, $\{i = 1, 2, \ldots, NP\}$, the corresponding mutant vector $\mathbf{v}$ is generated as follows:

$$\mathbf{v} = \mathbf{x}_{r0,g} + F \cdot (\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}), \quad r_0, r_1, r_2 \in \{1, 2, \ldots, NP\}. \tag{1}$$

where $\mathbf{x}_{r0,g}$ is a base vector and the indexes satisfy $r_0 \neq r_1 \neq r_2 \neq i$. It is obvious that at least four individuals are needed to implement the above mutation operation, implying that $NP \geq 4$. The scaling factor F lies within the range (0, 2), usually less than 1 [22].

2.2. Crossover

After mutation, the target vector (individual) is mixed with the mutant vector, and the following crossover operation is used to form the final trial vector:

$$\mathbf{u}_{i,g,j} = \begin{cases} \mathbf{v}_{i,g,j}, & \text{if } rand(0, 1) \leq CR \text{ or } j = j_{rand} \\ \mathbf{x}_{i,g,j}, & \text{otherwise} \end{cases}, \tag{2}$$

where $i = 1, 2, \ldots, NP$ and $j = 1, 2, \ldots, D$. $CR \in (0, 1)$ is the crossover rate that controls the probability of creating components for the trial vector $\mathbf{u}$ from the mutant vector $\mathbf{v}$. The index j_{rand} is an integer randomly chosen from the set $\{1, 2, \ldots, NP\}$, ensuring that at least one component of the trial vector is provided by the mutated vector.