Applied Soft Computing 34 (2015) 286–300

Contents lists available at ScienceDirect

Applied Soft Computing

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

Review article

Distributed evolutionary algorithms and their models: A survey of the state-of-the-art[☆]

Yue-Jiao Gong[a,b], Wei-Neng Chen[a], Zhi-Hui Zhan[a], Jun Zhang[a,c,*], Yun Li[d], Qingfu Zhang[e], Jing-Jing Li[f]

[a] School of Advanced Computing, Sun Yat-Sen University, Guangzhou, China
[b] Faculty of Science and Technology, University of Macau, Macau
[c] State Key Laboratory of Mathematical Engineering and Advanced Computing, China
[d] School of Engineering, University of Glasgow, UK
[e] School of Computer Science and Electronic Engineering, University of Essex, UK
[f] School of Computer Science, South China Normal University, China

ARTICLE INFO

Article history:
Received 20 December 2014
Received in revised form 29 April 2015
Accepted 30 April 2015
Available online 11 May 2015

Keywords:
Distributed evolutionary computation
Coevolutionary computation
Evolutionary algorithms
Global optimization
Multiobjective optimization

ABSTRACT

The increasing complexity of real-world optimization problems raises new challenges to evolutionary computation. Responding to these challenges, distributed evolutionary computation has received considerable attention over the past decade. This article provides a comprehensive survey of the state-of-the-art distributed evolutionary algorithms and models, which have been classified into two groups according to their task division mechanism. Population-distributed models are presented with master-slave, island, cellular, hierarchical, and pool architectures, which parallelize an evolution task at population, individual, or operation levels. Dimension-distributed models include coevolution and multi-agent models, which focus on dimension reduction. Insights into the models, such as synchronization, homogeneity, communication, topology, speedup, advantages and disadvantages are also presented and discussed. The study of these models helps guide future development of different and/or improved algorithms. Also highlighted are recent hotspots in this area, including the cloud and MapReduce-based implementations, GPU and CUDA-based implementations, distributed evolutionary multiobjective optimization, and real-world applications. Further, a number of future research directions have been discussed, with a conclusion that the development of distributed evolutionary computation will continue to flourish.

Contents

☆ This work was supported in part by the National High-Technology Research and Development Program (863 Program) of China No. 2013AA01A212, in part by the NSFC for Distinguished Young Scholars No. 61125205, in part by the NSFC Key Project No. 61332002, and in part by the NSFC Joint Fund with Guangdong under Key Projects No. U1201258 and No. U1135005, and in part by the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing.

* Corresponding author at: School of Advanced Computing, Sun Yat-Sen University, Guangzhou, China. Tel.: +86 13570277588.
E-mail address: junzhang@ieee.org (J. Zhang).

http://dx.doi.org/10.1016/j.asoc.2015.04.061
1568-4946/© 2015 Elsevier B.V. All rights reserved.

1. Introduction

With metaheuristic and stochastic characteristics, evolutionary computation (EC) has shown to be effective solvers for hard optimization problems in real-world applications. However, with rapid development of the information age and the emergence of "big data", the increasing size and complexity of the problems has posed new challenges to EC. This is especially so if the search space involves a huge number of local optima or the computational cost of fitness evaluation becomes extremely high. When a traditional sequential evolutionary algorithm (EA) is unable to provide satisfactory results within a reasonable time, a distributed EA (dEA), which deploys the population on distributed systems, can improve the availability. It also offers an opportunity to solve extremely high dimensional problems through distributed coevolution using a divide-and-conquer mechanism. Further, the distributed environment allows a dEA to maintain population diversity, thereby avoiding local optima and also facilitating multiobjective search.

The framework of developing a distributed EA is illustrated in Fig. 1. Its fundamental algorithms embrace all kinds of EAs including the genetic algorithm (GA), evolutionary programming (EP), evolution strategy (ES), genetic programming (GP), and differential evolution (DE). Moreover, other population-based algorithms, such as ant colony optimization (ACO) and particle swarm optimization (PSO), share common features with EAs and are hence also included in this survey. Then, by employing different distributed models to parallelize the processing of EAs, various dEAs can be designed. The logistical distributed models have several issues to address, such as the distribution of evolution tasks and the protocols for communications among processors. The granularity of the distribution may be at the population level, the individual level, the operator level, or the variable level. Correspondingly, there can be various communication rules in terms of the content, frequency, and direction of message passing. In the literature, master-slave [31], island (a.k.a. coarse-grained model) [56,99], and cellular (a.k.a. fine-grained model) [51,1] models have been commonly used to build dEAs. Moreover, other models such as the hierarchy (a.k.a. hybrid model) [41], pool [104], coevolution [121,122], and multi-agent models [10] are also widely accepted. After designing a dEA, different programming languages and tool sets can be adopted to implement the algorithm, such as the Message-Passing Interface (MPI) [63], MapReduce [81], and Java [129,38]. There also exist software packages for dEC, such as the Paladin-DEC [126,127] and ParadisEO [14,15]. Finally, the format of the physical platform that can be used to deploy the algorithms includes cluster [73], grid [39], P2P network [141], cloud [44], and GPU [151]. These platforms have different architectures, network connectivity, resource management schemes, and operating systems. Two recent papers, [133,65], review and discuss the parallel and distributed GAs in considering different physical platforms. The selection of the underlying platform partially influences the implementations of dEA models, and also determines the system performance such as scalability and fault-tolerance.

As there exist a very large number of research outputs in dEAs, it is impossible to cover all the relevant works within the page limit of this article. Therefore, references are presented based on their influence, rigor, years of publication, numbers of citations, and coverage. Models (the second layer in Fig. 1) continue to be a focus of interest in developing dEAs and will hence form the main body of this article.

We aim at providing readers with an updated, comprehensive and systematic coverage of dEAs and the state-of-the-art dEA models. The characteristics (or novelties) of this article are presented as follows. (1) Compared with [3,16,4,131] published ten years ago, this survey introduces and describes more recent works in this area. In addition to the master-slave, island, cellular, and hierarchical models surveyed in the literature [3,16,4,131,97], we further review some state-of-the-art distributed models for EC, including resource pool-based model, coevolution model, and multi-agent model. To the best of our knowledge, no previous survey of dEC covers these fields. (2) To update with a systematic treatment on the research progress, we semantically divide dEA models into two major categories, i.e., population-distributed models and dimension-distributed models. The operating mechanisms of different dEA models are analyzed and compared, as well as their corresponding performance, advantages, disadvantages, and ranges of applicability. (3) Recent research hotspots, including cloud and MapReduce-based implementations, GPU and CUDA-based implementations, multiobjective dEAs, and real-world applications, are also discussed in this survey. (4) In addition to a literature review, emerging research directions and applications are presented for possible further development.

The rest of this article is organized as follows. Section 2 introduces terminologies for a systematic treatment and classification. Section 3 presents population-distributed and dimension-distributed models, followed by a summary and analysis of characteristics in Section 4. Section 5 is devoted to the four recent research hotspots. Finally, we highlight some potential future directions in Section 6 and draw conclusions in Section 7.

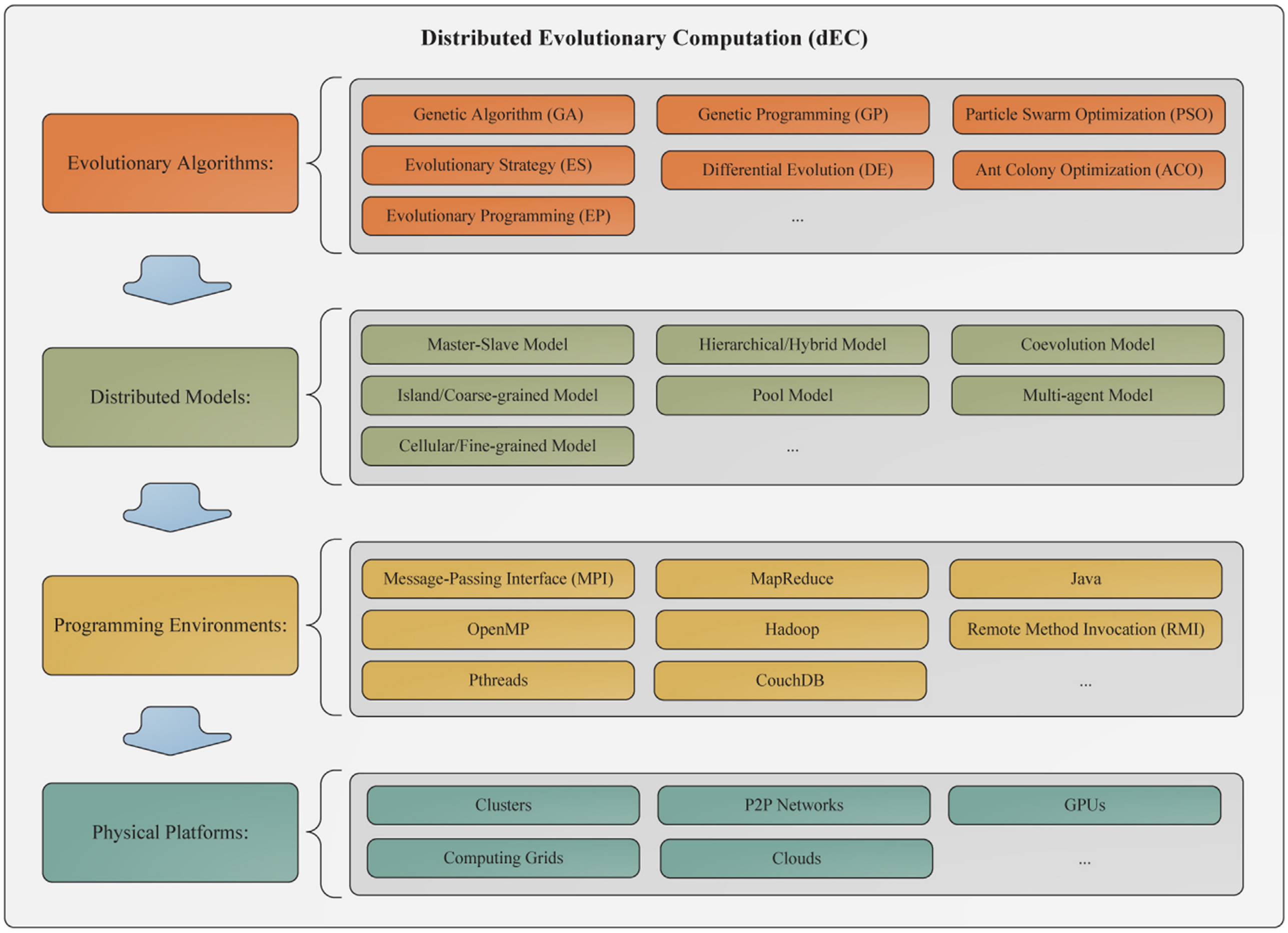

Distributed Evolutionary Computation (dEC)
Evolutionary Algorithms:
Genetic Algorithm (GA)
Genetic Programming (GP)
Particle Swarm Optimization (PSO)
Evolutionary Strategy (ES)
Differential Evolution (DE)
Ant Colony Optimization (ACO)
Evolutionary Programming (EP)
...
Distributed Models:
Master-Slave Model
Hierarchical/Hybrid Model
Coevolution Model
Island/Coarse-grained Model
Pool Model
Multi-agent Model
Cellular/Fine-grained Model
...
Programming Environments:
Message-Passing Interface (MPI)
MapReduce
Java
OpenMP
Hadoop
Remote Method Invocation (RMI)
Pthreads
CouchDB
...
Physical Platforms:
Clusters
P2P Networks
GPUs
Computing Grids
Clouds
...

Fig. 1. The general dEC framework.

2. Terminologies

In this section, we briefly introduce the terminologies that will be used throughout this article. The first two concepts, i.e., "synchronism/asynchrony" and "homogeneity/heterogeneity", are widely used to describe the properties of dEAs, whereas the "speedup", "distributed efficiency", "fault-tolerance", and "scalability" are performance metrics for evaluating dEAs.

Synchronism and asynchrony. An indispensable issue in a dEA as well as any other distributed algorithm is the communications among processors. If all communications are controlled by a clock signal, then the algorithm is said to be synchronous, otherwise asynchronous. In an asynchronous dEA, communications take place more freely or automatically driven by data.

Homogeneity and heterogeneity. For dEAs, homogeneity and heterogeneity are used to describe whether the evolution tasks on different processors are of the same settings. In a homogeneous dEA, each processor adopts the same operators, control parameters, fitness evaluation, etc., whereas in a heterogeneous dEA, the local algorithmic settings for different processors vary.

Speedup and distributed efficiency. The distributed processing performance of dEAs is qualified by a speedup measure, the ratio of sequential execution time to parallel execution time of the algorithm [31]. Ideally, the speedup should be equal to the number of processors being used. Based on this, distributed efficiency is defined as the ratio of speedup to the number of processors and its ideal value is 100%. In practice, the speedup and efficiency of dEAs may be limited by the computational overhead, the performance of the most loaded node, and the communication speed between processors.

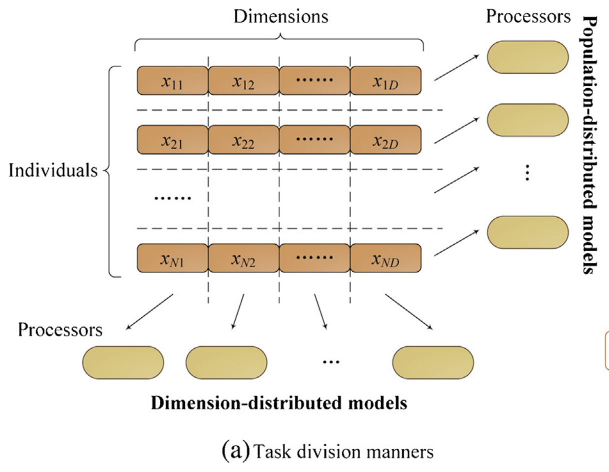
Fault-tolerance. When running EAs on a physical distributed system, part of the underlying hardware or network may encounter failure. Fault-tolerance measures the ability of a dEA to continue optimization in the condition of some physical components failing. A fault-tolerant dEA will not be suspended in such condition, instead, it continues search with the remaining working nodes at a level of graceful degradation.

Scalability. The scalability of dEAs involves two aspects: "size scalability" and "task scalability". Size scalability refers to the ability of the algorithm to achieve proportionally increased performance by increasing the number of processors. Task scalability refers to the ability of algorithm to adapt to the changes in the problem scale, e.g., whether the algorithm can retain its efficiency when the problem dimension increases.

As a final note, within this paper, the terms dEC and dEAs are used in a general sense, which include both the algorithms implemented on parallel systems (where the processors or threads are tightly coupled with a shared memory) and the algorithms implemented on distributed systems (where the processors are loosely coupled with a computer network).

3. Models of distributed evolutionary algorithms

Basically, a distributed EA divides computing tasks based on two types of models. As illustrated in Fig. 2(a), a "population-distributed" model distributes individuals of the population (or



Dimensions
Processors
Population-distributed models
x_{11} x_{12} …… x_{1D}
x_{21} x_{22} …… x_{2D}
x_{N1} x_{N2} …… x_{ND}
Individuals
Processors
Dimension-distributed models
(a) Task division manners

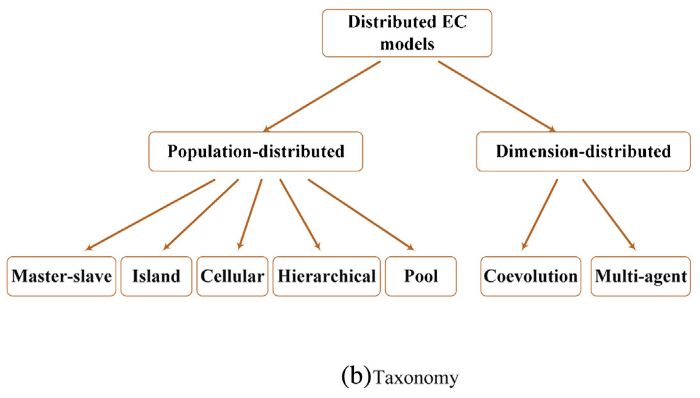


Fig. 2. Classification of the "population-distributed" and "dimension-distributed" models: (a) task division manners; (b) taxonomy.

subpopulations) to multiple processors or computing nodes, whilst a "dimension-distributed" model distributes partitions of the problem dimensions (or subspaces). The population-distributed model can be further divided to master-slave [31], island (a.k.a. coarse-grained model) [56,99], cellular (a.k.a. fine-grained model) [51,1], hierarchical (a.k.a. hybrid model) [41], and pool models [104], as illustrated in Fig. 2(b). On the other side, the dimension-distributed model can be divided to coevolution [121,122] and multi-agent models [10].

3.1. Master-slave model

The master-slave model summarizes a distributed approach to the EA operations and domain evaluations as illustrated in Fig. 3. The master performs crossover, mutation, and selection operations, but sends individuals to slaves for fitness evaluations because these form the majority of the computing load. As the evaluations of individuals are mutually independent, there is no requirement of communication among slaves. The master-slave model is hence simple, in which communications only occur when the unique master sends individuals to slaves and the slaves return the corresponding fitness values back to the master in each generation.

Variants to improve efficiency. For problems whose evaluation costs are not relatively high, however, employing a master-slave model may become inefficient in that communications occupy a large proportion of time in the dEA. In recent years, variants of master-slave dEAs have been developed to address this issue. A commonly used method is to distribute not only the evaluation tasks but also the individual update tasks to slave nodes [63,84,105,61]. Another approach is a coarse-grained master-slave model in which each slave processor contains a subpopulation, while the master receives the best individual from each slave and sends the global best information to all slaves [144]. Note that such

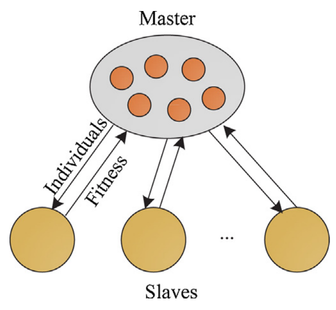


Fig. 3. Illustration of master-slave EAs.

a coarse-grained master-slave model is different from the island model being introduced in the next subsection. First, the control of the former is centralized whereas the control of an island model can be either centralized or distributed. Second, as mentioned above, slaves do not communicate with slaves, but in an island model, the islands frequently communicate with each other. The third possible way to improve the distributed efficiency of master-slave dEAs is to conduct local search on slaves [147,119]. In the algorithms, master conducts basic EA for global search whereas the slaves execute local search by considering the individuals received from the master as neighborhood centers.

Synchronism and asynchrony. Most existing master-slave dEAs are synchronous that the master stops and waits to receive the information of all slaves before proceeding to the next generation. Some are asynchronous, where the selection operations on the master node perform on a fraction of the population only [106,107]. In an experimental study of a master-slave PSO algorithm [110], it is shown that synchronization plays a vital role in algorithm performance on load-balanced problems, whilst asynchronous dEAs are more efficient for load-imbalanced problems.

Speedup. The speedup and efficiency of master-slave dEAs may be limited by the master's performance and by the communication speed between master and slaves [17]. Specifically, the limitation is determined by the computational costs of the tasks executed on the slaves. For example, Dubreuil et al. [31] show that the master-slave model can perform well as long as the individual evaluation time is much greater than the message passing time, as can be expected. In their experiment, solving a problem requiring 0.25 s for evaluation yields an efficiency of 82%, but if the evaluation time increases to 1 s while the communication overhead remains the same, the efficiency becomes 95%.

Fault-tolerance. For massive dEAs, how to improve the fault-tolerance is another important issue. Gonzalez and De Vega [55] argue that master-slave dEAs are intrinsically fault-tolerant. In [120], a fault-tolerant DE algorithm based on a master-slave model is proposed, where the individuals are distributed to a grid of nodes for fitness evaluations and if certain individuals fail to return from their nodes in an acceptable time, they can be replaced with random individuals. This mechanism not only shows fault-tolerance, but can also help improve population diversity.

3.2. Island model

An island model, as well as a cellular model, is a spatially distributed model. The difference between an island model and

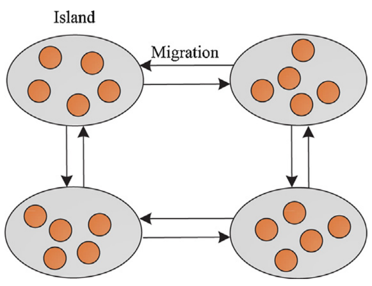


Fig. 4. Illustration of island EAs.

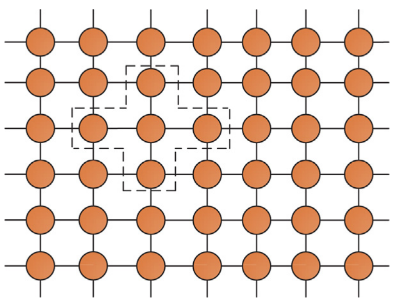


Fig. 5. Illustration of cellular EAs.

a cellular model lies in the parallelization grain. As depicted in Fig. 4, an island model is coarse-grained, where the global population is divided into several subpopulations, each of which is processed by one processor. Communications between the islands occur when certain individuals in one island migrate to another at a set interval. The migration mechanism includes the migration frequency and extent, the selection policy on the source island, and the replacement policy on the target island.

Synchronism and asynchrony. An island dEA is often synchronous that the best individual on each island propagates to all the other islands at a specific interval of generation [89,100,148]. Asynchronous island models also exist [30,80], where an island can receive migrated information as soon as it is ready. In general, synchronous island dEAs are simpler to design and implement, whereas asynchronous algorithms are more flexible and can maximize efficiency.

Homogeneity and heterogeneity. In a homogeneous island model, each subpopulation adopts the same settings of operators, control parameters, and fitness evaluations, etc. There exist two shortcomings. First, if the underlying physical system is heterogeneous, slower processors will hinder the efficiency of the algorithms. Second, using the same algorithmic settings on different subpopulations may not balance global exploration and local exploitation. Because of these, heterogeneous island models are developed. One example is the heterogeneous island GA [112], where subpopulations are arranged in a three-layer hierarchal manner: the top layer refines exploitation, the intermediate layer balances exploration and exploitation, and the bottom layer conducts full-on exploration. A hypercube island model is developed in [56], where subpopulations on the front side use different crossover operators for exploration and the others on the rear side adopt crossover operators that are more suitable for exploitation. Moreover, the exploration and exploitation degrees of subpopulations on the same side are gradual. Other heterogeneous island dEAs can be found in [138,139].

Topology and migration strategy. The original island dEAs adopt a complete graph as the topology of the islands (i.e., they are fully connected "using no topology"). Whitley and Starkweather [140] and Lorion et al. [79] put forward that, if the migration conducts among all islands, the distributed algorithm has almost the same search behavior as a sequential algorithm. In recent years, research into network topology of island models has attracted much attention [140,79,148]. In [62], island DE algorithms with different network topologies including ring, torus, hypercube, and hierarchy are studied, and experimental results confirm the superior performance of adopting a network topology in island DE.

The advantages of using an island model include not only time saving, but also the improvement of global search ability of EAs. Traditional EAs with a single population suffer from premature convergence problem when all individuals gather in a same valley. By deploying a number of subpopulations on isolated islands, it is possible to maintain more than one best individual (attractor). Within the time interval between communications, individuals on different islands can evolve with different directions. This helps EAs to maintain population diversity so as to repel local optimality [99,76]. In [7], an island-based distributed DE algorithm is tested on the well-known CEC 2005 test suite for real-parameter optimization [123], with results showing that the algorithm outperformed eight sequential EAs. Further, the work in [62] shows a tradeoff between the exploration and convergence feature of DE by using different migration frequencies. With a higher communication frequency, the island DE can converge faster but may get trapped, and, oppositely, the algorithm exhibits better global exploration ability but converges much slower when the communication frequency is lower. Moreover, the work in [130] indicates that the migration extent also bears a significant impact on the algorithm performance. In [18], a Markov chain model of predicting the expected solution quality of dEAs is developed, with correctness verified by numerical experiments.

Scalability and fault-tolerance. Regarding the system performance of island model, Hidalgo and Fernandez [58] argue that, as performance of island-based dEAs is highly sensitive to the number of islands used and the resulting granularity, scalability of the island model can be limited. Besides, Hidalgo et al. [59] point out that, to a certain extent, fault-tolerance also exists in an island model.

3.3. Cellular model

Illustrated in Fig. 5, a cellular model is fine-grained and spatially structured, which has only one population but arranges the individuals on the grid, ideally one per processor (cell). The interaction among individuals is realized through the communication defined by a network topology. Each individual can only compete and mate within its neighborhood. As the neighborhood of individuals overlaps, good individuals can propagate to the entire population.

Synchronism and asynchrony. Similar to an island EA, a cellular EA (cEA) can also be either synchronous or asynchronous [132]. In the former, all cells update their individuals simultaneously, whereas in the latter, the cells are updated one by one. The four commonly used asynchronous update strategies are the fixed line sweep (LS), fixed random sweep (FRS), new random sweep (NRS), and uniform choice (UC), as proposed in [109]. Alba et al. [2] compare the asynchronous cEAs using these four update strategies with synchronous cEAs on both discrete and continuous problems. Their experimental results show that, with respect to discrete problems, asynchronous algorithms are more efficient, but synchronous algorithms can achieve better fitness. On the contrary, in solving continuous problems, they draw complementary conclusions that asynchronous cEAs are better in solution quality whereas synchronous cEAs win in efficiency. A novel asynchronous communication method for a cEA is proposed in [71], which uses self-adaptation of the migration rate to provide a better leverage network capacity than using a fixed migration rate.

Topology. So far, most efforts in cEAs have been devoted to analyzing the effects of different topologies on the algorithm performance. In particular, the selection intensity in cEAs on various topologies has been widely investigated. In [51,47,49,46], Giacobini et al. study the selection intensity of cEAs with linear topology, toroid topology, and regular lattices as well as the asynchronous cEAs, respectively. In their studies, a takeover time measure proposed by Goldberg and Deb [53] is used. The takeover time is defined as the duration of a single individual propagating to the entire population with no variation other than selection. The shorter the takeover time is, the higher the selection intensity is, which represents a higher exploitation degree of the algorithm. Their experiments show that choosing of a network topology can have significant impact on the selection intensity and the algorithm performance.

In recent years, as the network scale of cEAs becomes larger, complex networks such as the well-known small-world network and scare-free network have been introduced to cEAs. In [50,48], Giacobini et al. use takeover time analysis to investigate the selection intensity of cEAs based on small-world topology and scale-free topology, respectively. In [68], the performance of cEAs using four topologies, including the 2D regular lattice, small-world network, random graph, and scale-free network, is investigated. Their experimental results show that, with the increase of the problem complexity, the ideal topology should change from one with a high mean degree distribution (the regular topologies) to a network with a high clustering coefficient (the complex networks).

Apart from the takeover time, a ratio measure of the neighborhood radius to the topology radius proposed in [108] has been widely used to study the performance of cEAs. In [5], Alba and Troya conduct a set of tests to analyze the performance of cEAs with different ratio values on different classes of problems. The paper concludes that a cEA with low ratio is more effetive for multimodal and/or epistatic problems, whereas a cEA with high ratio performs better on non-epistatic and simple problems. Based on these, a novel cEA with dynamic ratio from low to high during a run is developed, which is verified to be efficient in the paper. Further, an adaptive cellular GA is developed in [1], which adaptively adjusts the neighborhood-to-topology ratio during the search process according to some rules defined on the average fitness (AF), population entropy (PH), and their combination (AF + PH).

3.4. Hierarchical model

The hierarchical model, also known as hybrid model, combines two (or more) distributed models hierarchically to take advantages of both models for improving scalability and problem-solving capability.

Island – master-slave hybrid. In [12,13,75], the population is divided into several subpopulations, which run on different master processors and communicate in some specific time. For each subpopulation, the master sends the individual evaluation tasks to its own slave processors so as to further improve parallelization grain. As shown in Fig. 6(a), the model is island and master-slave hybrid, which uses island model in upper layer and master-slave model in lower layer. Such a model not only eases scalability limitation of an island model but also reduces dependency of the single master node in a master-slave model. In [13], Burczyski et al. show that the speedup of their island–master-slave hierarchical algorithm is relatively linear.

Island – cellular hybrid. The hybridization of island and cellular models has also attracted attention. Folino and Spezzano [42] develop a distributed GP algorithm running on multiple islands that contain local cellular GP approaches. Such a model is shown in Fig. 6(b), where an island and a cellular model are adopted in the upper and lower layers, respectively. Numerical results on benchmark functions show that a hierarchical GP algorithm presents accuracy comparable with classical distributed models, while providing advantages of high scalability and fault-tolerance [42]. The algorithm has been further improved by Folino *et al.* and applied in pattern classification in [41].

Island – island hybrid. Another hierarchical model of dEAs is to adopt the island models in both upper and lower layers, as shown in Fig. 6(c). Herrera et al. [57] point out that, in this kind of model, a key issue is to develop two different migration approaches, i.e., local and global ones, since they establish the real hierarchy between basic dEAs and the hierarchical dEAs. Moreover, the advantages of using such a hierarchical model include improved efficiency of each node, more diverse collaboration, and good conjunction of homogeneous and heterogeneous dEAs. Based on these, Herrera et al. develop a heterogeneous hierarchical dEA and achieve promising results.

3.5. Pool model

The above master-slave, island, cellular, and hierarchical models offer the promise of massive scalability and fault-tolerance if the problem to solve can be properly adapted to their size and peculiarities [85]. However, there is still certain inflexibility and inefficiency that hinders the use of these models. For example, in a master-slave model, with the increase of the number of slave nodes, the speedup will eventually become poor when the master saturates. In island and cellular models, the predefined topology and the rigid connectivity restrict the amount of islands or cells to be used and the spontaneous cooperation among the nodes. Although the models can be asynchronous and heterogeneous, the asynchronization and heterogeneity pose restriction on the performance of corresponding dEAs. Compared with this, a pool model deploys a set of autonomous processors working on a shared resource pool. The processors are loosely coupled, which do not know each other's existence and interact with only the pool. The model provides a natural approach to realizing asynchronization and heterogeneity.

Instance. For better understanding of the pool model, we describe a distributed pool architecture for EC proposed by Roy *et al.* [104] in detail as an instance. As illustrated in Fig. 7, the pool is a shared global array of length n representing n individuals in the population. Then, the array is partitioned into p segments of size u, which correspond to p processors (or threads). Each processor can read individuals from any segments of the array, but can only write the individuals back to its own partition. In the optimization process, a processor randomly chooses u individuals from the entire pool to undergo genetic operations. After generating u offsprings $c_1, c_2, \ldots, c_u$, the processor writes each new individual c_i back to the ith entry of its own partition if the fitness of c_i is better than that of the current ith entry. In summary, key issues of designing such a dEA include 1) implementing the resource pool, 2) individual selection policy (consuming policy on the pool), and 3) individual replacement policy (producing policy on the pool).

Advantages. As processors are loosely coupled to work on a shared resource pool, they can accommodate asynchronization and heterogeneity relatively easily. Moreover, in a pool model, the set of participating processors can be dynamically changed, and the system works well even when some of the processors crash. By replicating (backing up) the resource pool, the model can achieve superior fault-tolerance. Another possible advantage of such a loosely coupled distributed model is that it can be cost-efficient. For example, volunteers around the world can contribute the idle time of their computers for processing the tasks.

Resource pool. In a pool-based distributed model for EAs, how to implement the resource pool is a crucial issue to address. TupleSpace (TS), the shared-memory programming model of Linda,

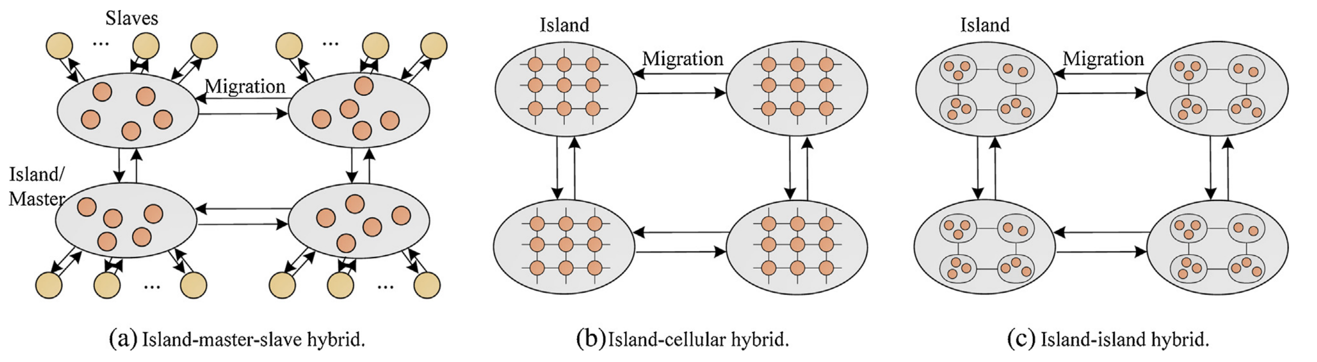

Slaves
Migration
Island/
Master
(a) Island-master-slave hybrid.
Island
Migration
(b)Island-cellular hybrid.
Island
Migration
(c) Island-island hybrid.

Fig. 6. Illustration of hierarchical EAs: (a) Island–master-slave hybrid; (b) island-cellular hybrid; (c) island–island hybrid.

provides a virtual shared-memory data storage that processors can read and write. By mapping a GA onto TS, a pool-based distributed GA is built, with a natural load-balancing effect that faster processors end up doing more work than slower processors [23]. This work is perhaps the first dEA based on a pool model. Since then, work has been reported on employing a database as the central resource pool for dEAs. There are two major advantages of adopting a database as the pool. First, as suggested by Bollini and Piastra [9], an object-oriented database management system provides mature transaction and data locking mechanisms. It allows any number of evolutionary processes run in parallel on the underlying population without extra control policies. Second, the database can persistently and permanently store the population until it is modified by the users. Therefore, the computation of dEAs can span weeks or even months, such as the distributed BEAGLE proposed in [43].

In recent years, there are many pool dEAs developed, based on matching implementations of EAs to programming models or toolkits such as MapReduce and CouchDB. As this section mainly focuses on the models rather than the implementations, these works will be described briefly in Section 5.1.

3.6. Coevolution model

A coevolution model is a dimension-distributed model. Instead of dividing the population, a dimension-distributed model divides a high dimensional complex problem into several lower dimensional and hence simpler problems. Note that, however, dimension distributed and population-distributed models have no clear boundaries, and a dimension-distributed model can also arrange its tasks in an island, cellular, or hierarchical manner, etc.

If the problem is decomposable, i.e., the sub-problems can be solved independently, the subcomponent on each processor can evolve without interacting with the others. At the end of the optimization, by jointing the sub-solutions together, an optimal solution of the entire problem emerges. Unfortunately, most of practical optimization problems exhibit complex interdependencies, for which the solution obtained by the above divide-conquer-and-joint mechanism may be inferior. It is suggested that a change of one subcomponent (e.g., a new optimal solution found in one processor) can lead to the deformation or warping of the fitness landscapes in its interdependent subcomponents. The distributed coevolution model is developed to deal with the above problem.

In biology, coevolution indicates that the change of a species triggers the change of its related species, and then leads to an adaptive change of its own part, and so forth. This way, different species in the environment have correlative dependence, and, from a general viewpoint, they evolve cooperatively. The coevolution model for dEC borrows this concept, where each node performs a local evolution process in a solution subspace. Then, by intercommunication, the nodes interact with the others, adaptively adjust their search direction, and cooperatively find the global optimum. Potter and De Jong [101] point out that, when developing coevolutionary algorithms, four issues need to be addressed. They are problem decomposition, the evolution of interdependent subcomponents, credit assignment (evaluation), and maintenance of diversity.

Fundamental framework. In 2004, Subbu and Sanderson [121,122] develop a fundamental framework for distributed coevolutionary algorithms, analyze the convergence of the framework, and examine the network-based performance. As illustrated in Fig. 8, assuming the variable vector x consisting of p blocks (x_1, x_2, ..., x_p), each node i in the algorithm performs a local evolutionary search process by considering the ith block x_i primarily and the other $p-1$ blocks secondly. Specifically, the local reproduce operation is conducted on the primary block x_i while the remaining variables are clamped. In the evaluation, the fitness of the whole solution (including both the primary and secondary blocks) is calculated, and the local algorithm is more likely to preserve solutions with better fitness. In this way, the primary variable block on the node evolves. Then, in the intercommunication phase, the nodes update their secondary variable blocks. By alternating the

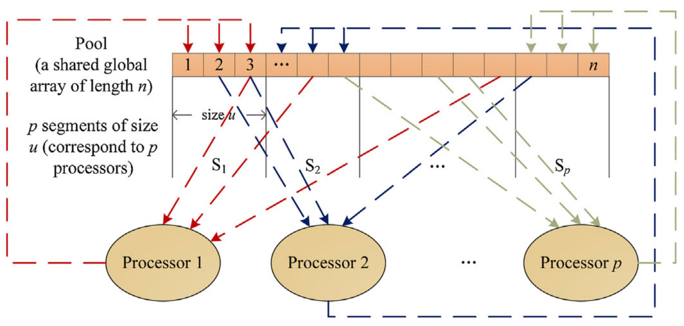

Pool
(a shared global array of length n)
p segments of size u (correspond to p processors)
1
2
3
...
n
size u
S1
S2
...
Sp
Processor 1
Processor 2
...
Processor p

Fig. 7. Illustration of resource pool-based EAs.

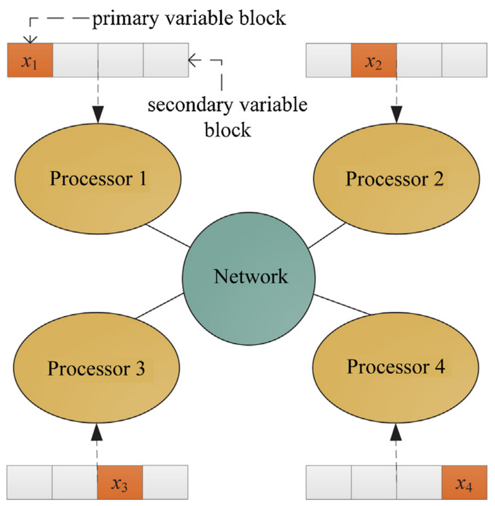


Fig. 8. Illustration of distributed coevolutionary algorithms.

local search and intercommunication phases, an adaptive system is built, capable of solving high-dimensional complex problems. Most recent distributed coevolutionary algorithms have adopted the above framework, but differ in problem decomposition strategies, local EAs, and intercommunication.

Decomposition strategy. It may be possible to decompose an n-dimensional problem into n one-dimensional problems in some applications. However, this is often not the case and hence Yang et al. [145] suggest using a group-based decomposition strategy to better capture variable interdependencies for nonseparable problems. For this, an adaptive weighting strategy is developed in [145], where the chance of one dimension to be assigned into a subcomponent is adaptively adjusted during the search process. Li and Yao [72] further improve the decomposition strategy of Yang et al. by dynamically changing the group size, and successfully solve up to 2000-dimensions problems. In [73], a coevolutionary DE is designed for power system optimization. The whole system is decomposed into a series of subsystems with different numbers of control variables by using an agglomerative hierarchical clustering (AHC) method. Each species is responsible for the regulation of control variables in its own subsystem, while taking the values of the other control variables from the global best individual found so far. Ray and Yao [102] develop an adaptive variable partition strategy, in which all variables involve together at the beginning of the algorithm and then be grouped by a correlation coefficient.

Intercommunication and credit assignment. Intercommunications in [149] are realized through adaptive migration of the best primary variable block of each node during the optimization. Potter and De Jong [101] and Tan et al. [128] point out that combining the primary block of one species with only the best blocks from the other species is often too greedy, which may result in getting trapped in local optima. In their proposed algorithms, the primary block of each species is first combined with the best blocks from other species and then combined with some random representatives of every other species. After evaluation, the better one is retained. In [52], Goh and Tan further introduce a competitive process in the coevolution to improve the contribution of each species. In their proposed competitive-cooperative coevolutionary paradigm, the interplay of competition and cooperation facilitates the discovery of interdependencies among species.

3.7. Multi-agent model

In the above coevolution model, the global goal of the entire system is essentially the local goal of each subcomponent, which is achieved by coordination of subpopulations. In comparison, a multi-agent model does not require any direct coordination of agents to achieve the global goal. Instead, it adopts game-theoretic method in the field of distributed artificial intelligence (DAI) that agents optimize local functions and establish some equilibrium. In the equilibrium, once the local objectives cannot be further improved, the global goal of the entire system is achieved. In this way, the global goal is realized by observation rather than evaluation.

Methodology. The main idea of a multi-agent model is to consider a dEA as a multi-agent system that p players (agents) are playing a strategy game. Each player in the game has a payoff function that depends on the actions of itself and its limited neighbors. Then, each player plays independently in the game and selects actions to maximize its own payoff selfishly. A widely accepted solution for this non-cooperative game is the Nash equilibrium point, a p-tuple of actions for all players that anyone who deviates from it cannot improve its own payoff [96]. One issue to be addressed here is how to convert the global optimal solutions of the problem (or the maximal price points in the game) into Nash equilibrium points. Not all practical problems can be solved by dEAs based on a multi-agent model, unless the problem accommodates the above converting process.

Loosely coupled GA. EA based on a multi-agent model appeared as early in the 1990s as the loosely coupled GA (LCGA) proposed by Seredynski [113]. In this algorithm, each player creates a subpopulation of its actions, and the payoff is considered as the evaluation value of the local fitness function. Standard genetic operations, including selection, crossover, and mutation, are applied locally to the subpopulation of actions. Then, after a number of iterations, the players find actions corresponding to the Nash equilibrium. Experimental study in [113] has shown that the LCGA can optimize the global objective in a fully distributed way of evaluating only local fitness functions. Afterwards, the LCGA is widely used in both function optimization and real-world applications such as mapping and scheduling problems [10,114].

Comparisons between multi-agent and coevolution models. A comprehensive comparison between the LCGA and the cooperative coevolutionary GA (CCGA) is reported in [115], which can also be regarded as a comparison between the multi-agent model and the coevolution model. As introduced above, the main difference between LCGA and CCGA is that the former evolves local objectives on agents and requires no coordination of agents. The experimental study in [115] illustrates that if the global objective problem can be expressed in a sum of local objectives, using LCGA is more efficient, as it can obtain high-quality solutions at a relatively low computational cost. However, for the other complex problems that are hard to be expressed in a fully distributed way, CCGA outperforms LCGA. The study in [22] shows similar conclusions, although Danoy et al. further point out that LCGA is more scalable than CCGA. Besides the LCGA, a multi-agent memetic algorithm named MA^2 is developed in [98], in which each agent in the multi-agent system is a subpopulation of a memetic algorithm (GA with local search). The algorithm has also shown its powerfulness in tackling high-dimensional optimization problems.

4. Summary and analysis

In this section, we summarize and analyze dEC models by comparing their parallelism, search behaviors, objectives, communication costs, scalability, and fault-tolerance, for the ease to readers in considering future work. This is conducted in a general sense regarding our above presentation of models. For example, if model A offers a higher scalability than model B, it implies that algorithms with model A “generally” offer a higher scalability than B,

Table 1
Comparisons of dEC models.

Model	Parallelism level	Objective function	Search behavior	Commun. cost	Scalability	Fault-tolerance
Master-slave	Operation, evaluation	Global	Similar to sequential EA	Medium to high	Medium	High
Island	Population	Global	Better diversity	Low to medium	Low	Medium
Cellular	Individual	Global	Better diversity	Medium	Medium to high	Medium to high
Hierarchical	Population, individual, operation	Global	Better diversity	Medium	Medium to high	Medium to high
Pool	Population, individual, operation	Global	Depending on algorithmic components	Low	High	High
Coevolution	Variable, variable-block	Global	Dimension reduction	Medium	Low	Low
Multi-agent	Variable, variable-block	Local	Dimension reduction	Low	Low	Low

but exceptions may exist in implementations. Further, in the comparisons, we assume that the five population-distributed models do not use problem decomposition, which is also in accordance with most reported work. The comparisons are summarized in Table 1 and are explained as follow.

4.1. Parallelism level

As a master-slave model parallelizes its individual evaluation tasks as well as some other operations (such as local search) on the slave nodes, the model has an operation-level of parallelism. Island and cellular models are population and individual level-based because they deploy subpopulations and individuals on the processors, respectively. As the hierarchical model can be island–master-slave hybrid, island-cellular hybrid, or island–island hybrid, etc., the parallelism level of the model can be operation-, individual-, and population-based. The two dimension-distributed models in Fig. 2(b) divide the evolution tasks by dimensions, where a model is variable-based if each processor engages with one variable only. Otherwise, if each processor optimizes a group of variables, the model has a variable block-based parallelism level.

4.2. Objective function

The first six models listed in Table 1 apply the unique global objective function to evaluate individuals on different processors. Differently, in multi-agent model, each processor has a local objective function to optimize. Nevertheless, it is to be noticed that the local objective-based multi-agent model can be implemented by different population-distributed models such as island, cellular, etc. In this sense, these population-distributed models can also have local objective functions on their parallel processors.

4.3. Search behavior

The search behavior of master-slave dEAs is similar to that of sequential EAs because it conducts the major evolution process of the algorithm on its master node and only sends some computationally expensive tasks to the slave nodes. For an island model, by deploying a number of subpopulations on isolated islands, the algorithm maintains more than one best individual (attractor) during the optimization and hence increases the population diversity. Literature also shows that using an island model not only saves computing time but also improves the global search ability of EAs [99,76,7]. For a cellular model, the use of local topologies reduces the selection intensity as well as the information propagation speed on the population network, which also results in a better population diversity. Besides, as a hierarchical model hybridizes the island model with others, it exhibits the effect of an increasing diversity as well.

The core method of a pool model is that the processors automatically evolve the individuals in the resource pool. The search behavior of the corresponding EAs is highly dependent on the algorithmic components in use (such as individual selection and replacement policies on the pool). Without specifying implementations of the algorithm, it is hard to identify the search behavior of a pool-based dEA. On the other hand, for the two dimension-distributed models with a divide-and-conquer method, the primary effect is the reduction of the problem space.

4.4. Communication cost

The entire computational cost of a dEA consists of three parts, namely, the evaluation cost C_{eval} of the problem, the operation cost C_{oper} of the baseline EA, and the communication cost C_{comm} of the distributed model. As C_{eval} and C_{oper} are relatively fixed, the communication cost C_{comm}, if significant, may affect the speedup and efficiency of the algorithm.

In a master-slave model, the communication cost is relatively high as the master frequently communicates with slaves sending individuals and receiving fitness values. Some variants, such as the coarse-grained master-slave model, decrease the frequency of communication between master and slaves and hence partially reduce the communication cost. For an island model, as the subpopulations share their information only at set intervals, the communication cost is relatively low. Nevertheless, the number of islands and the migration strategy can affect the communication cost to a great extent. In cellular and hierarchical models, communications occur between individuals, and hence the communication costs are considered to be medium. These are however highly related to the topology or hierarchical structure in use. In pool and multi-agent models, communication costs are relatively lower because no coordination among processors is needed. In a coevolution model where local search and intercommunication phases are alternated, the communication costs lie in a medium level.

4.5. Scalability

The scalability of a master-slave model is limited by the workload of the master node. When it is saturated, increasing the number of slaves would only decrease the distributed efficiency of the algorithm. For an island model, as the performance of the algorithm is sensitive to the number of islands used [58], the scalability is relatively low. In the literature, dEAs with island model always use a small number of processors. The scalability of a cellular model is better than an island model. With the introducing of the complex network-based topologies, the scalability of cellular models can be further improved.

Considering the hierarchical mode, as described in Section 3.4, it combines different models in a hierarchical fashion to improve its scalability. The pool model employs loosely coupled processors that do not know each other's existence. This offers high scalability. For the two dimension-distributed models, as the performance of the algorithms depends on the problem decomposition strategy (task division mechanism) used by the models to a great extent, the scalability of these two models is limited.

4.6. Fault-tolerance

The master-slave dEAs are fault-tolerant unless the master node fails. For island, cellular, and hierarchical models, failure of some processers will result in loss of some subpopulations or individuals. The fault-tolerance is medium to high. In a pool model, the set of participating processors can be dynamically changed, which enables the algorithms to achieve superior fault-tolerance. On the other hand, for the two dimension-distributed models, failure of a processor will result in losing subcomponents of the global solution and hence lead to a crash of the entire algorithm. Therefore, these two models are not fault-tolerant.

5. Recent research hotspots of dEAs

In this section, recent research hotspots of dEAs will be presented, including the cloud and MapReduce-based implementations, GPU and CUDA-based implementations, distributed multiobjective optimization, and some real-world applications. The work is however diverse, and hence this article is restricted to derivations, benefits and representative references.

5.1. Cloud and MapReduce-based implementations

Cloud computing represents a pool of virtualized computer resources. Compared to grid computing, the major difference is that cloud computing utilizes virtualization and autonomic computing techniques to realize dynamic resource allocations. As an on-demand computing paradigm, cloud offers high scalability and cost-effectiveness. Therefore, it is well suited to building highly scalable and cost-effective dEA systems for solving problems with requirements of variable demands. Although cluster [73], computing grid [39,28] and P2P network [141,71,111] have been widely used as physical platforms for dEAs, the studies of dEAs based on a cloud platform has received increasing attention since 2008 [44,45,27,16].

MapReduce is a programming model for accessing and processing of scalable data with parallel and distributed algorithms. Since introduced by Dean and Ghemawat [25] in 2004, MapReduce has been seen in various web-scale and cloud computing applications. The infrastructure of MapReduce provides detailed implementations of communications, load balancing, fault-tolerance, resource allocation, and file distribution, etc. All the things a user has to do are to implement the Map and the Reduce functions. In this way, the user can focus on the problem and algorithm only, without caring about the distributed implementation details. Because of this, implementing dEAs using MapReduce has attracted increasing attention in recent years [81,64,135,78,142,153,124].

Note that, although Google has described its MapReduce infrastructure, it has not released its system to public. Much of the work has been developed on Hadoop, a Java-based open-source clone of Google's private MapReduce infrastructure (by the Apache Lucene project). Moreover, Apache CouchDB, an open-source database, is used together with MapReduce to implement pool-based dEAs by Merelo et al. in [85,88,86,87].

Possessing many advantages, such as high scalability, cost-effectiveness, and transparency, the cloud and MapReduce-based implementations of dEAs still have some shortcomings. Generally, the speedup and distributed efficiency of dEAs deployed on clouds are lower than those deployed on clusters and computing grids, due to the increased communication overhead. The cloud computing paradigm prefers availability to efficiency, and hence the corresponding dEAs are more suitable for business and engineering applications, but rather the scientific computing where the speedup and distributed efficiency continue being a core index for performance evaluation.

5.2. GPU and CUDA-based implementations

A Graphics Processing Unit (GPU) is a powerful electronic circuit capable of executing hundreds of threads simultaneously. Early GPUs functioned as coprocessors to offload CPUs from tedious graphics tasks in video or game applications. As they are more efficient than CPUs, modern GPUs are not restricted to accelerate graphics or video coding, but used as a general-purpose processing unit for algorithms with intensive data processing tasks. With this trend, some recent research concentrates on implementing EAs on general-purpose GPUs (GPGPUs) to reduce the communication overhead and arrive at a high speedup. Numerous GPGPU-based EAs have thus been designed, with coverage of GA [117,82,83], GP [103], ES [155], EP [40], DE [136], PSO [151,91,154], and ACO [8]. Among these works, [40,136,103,83] apply master-slave model, [82,91] use island model, [117,155,8,154] adopt cellular model, and [151] applies hierarchical model.

It is to be noted that not all dEAs can benefit from being implemented on a GPU platform, but only the ones being synchronous, homogeneous, and lightweightly parallelized. The reasons are presented as follows. First, most GPGPU-based dEAs consider CPU and GPU as a host and a coprocessor, respectively, in which the data transferred between CPU and GPU are the population of individuals. The memory transfer process from CPU to GPU is commonly a synchronous operation, where the bus bandwidth and latency influence the performance significantly. Second, the "single program, multiple data (SPMD)" model of GPU device assumes that multiple processors execute the same program on different data (individuals in EAs). Thus, the distributed components of dEAs should contain the same operators. Third, as the thread of GPU is lightweight which can be considered as processing a data element, the task allocated to a thread by the EA should be in a very lightweight/fine-grained level. Although has some restrictions, a well-designed GPGPU-based EA can bring considerable speedup, e.g., "121×" when using 2014 threads and "286×" when using 15360 threads on the platform of Intel XeonTM E5420 CPU @2.5GHz, 2GB RAM, and nVidia GeForce GTX 280 GPU, as reported in [155].

Considering the programming environments, the Compute Unified Device Architecure (CUDA) developed by Nvidia is currently the most commonly used programming model to implement GPGPU-based EAs [136,8,155,91,154,103]. CUDA provides a sophisticated application programming interface (API) for an easy access of the "single instruction, multiple data (SIMD)" architeture. It builds a comprehensive environment to translate the C and C++ codes to the GPU platform, as well as Fortran, C#, Python, etc.

5.3. Distributed evolutionary multiobjective optimization

Unlike traditional single-objective problems (SOPs), a multiobjective optimization problem (MOP) involves multiple conflicting objectives with Pareto optimal solutions. MOPs are more difficult to solve than SOPs because the algorithms should be able to approximate a Pareto front instead of a single optimum. Because an EA is population-based, it is suitable to deal with a set of optimal solutions simultaneously in a single run. In order to characterize the entire Pareto front, a multiobjective EA (MOEA) employs a number of additional mechanisms, such as Pareto selection, solution maintenance, and diversity preservation. These mechanisms are often time consuming. The emergence of distributed MOEAs (dMOEAs) helps in speed and also provides a natural way to realize diversity preservation. In 2003, Veldhuizen et al. [134] paint a picture of dMOEAs with different paradigms, which leads dMOEAs to becoming one of the currently hottest research spots in the field of EC, dEC, and MOEAs.

Many dMOEAs are extensions of the Non-dominated Sorting GA (NSGA-II), a well-known MOEA proposed in [26]. Distributed NSGA-II with master-slave [35], island [39,11], and cellular models [69] can be found in the literature. Proposed in 2007, MOEA/D uses a decomposition method to transform an MOP into a set of SOPs to solve, which has remarkable performance in optimizing difficult MOP instances [150]. To make further improvement, parallel and distributed versions of MOEA/D are developed in [92,36,32,33]. Other dMOEAs include the distributed Strength Pareto EA (SPEA) [143], the distributed multiobjective PSO (MOPSO) [90], the distributed vector evaluated PSO (VEPSO) [137], and the parallel single front GA (PSFGA) [24], all of which are based on island models. In comparison, the dMOEAs proposed in [93,34] employ cellular models. In addition, Tan et al. [128] developed a distributed cooperative coevolutionary algorithm for multiobjective optimization, where the decision vectors are divided into subcomponents and evolved by cooperative subpopulations. By executing intercommunication of subpopulations residing in the distributed system and incorporating archiving, dynamic sharing, and extending operators, the algorithm is able to efficiently approximate solutions uniformly along the Pareto front. Other dMOEAs based on divide-and-conquer and coevolution techniques can be found in [29,152].

Although being a vibrant area, the research of dMOEAs still has some critical issues to be further addressed. Existing dMOEAs assume an ideal running environment that all processors are homogeneous and the communication costs between processors are identical, which is not always the case. The design of heterogeneous and asynchronous dMOEAs needs exploration. As described in Section 3.5, the resource pool-based model provides a natural way to realize asynchronization and heterogeneity. The model is very suitable for developing dMOEAs since we can deploy the searched nondominated solutions (or the so-called external archive) in the shared resource pool and let processors autonomously access and process them. This also brings more flexibility in designing the algorithms because it is now possible to assign heterogeneous tasks, such as individual reproduction, Pareto selection, solution maintenance, and diversity enhancement, to different sets of processors. Currently, the study of pool-based dMOEAs is still missing, which could be a potentially useful future direction. On the other hand, few efforts have been paid on regularizing the evaluation of dMOEAs such as proposing uniform test suites and performance metrics. Instead, the test suites and metrics of traditional MOEAs are applied, however, they are inadequate to investigate and analyze the performance of different dMOEAs, such as the scalability and speedup. To fully test the performance of dMOEAs, the test suite should cover a wide range of instances, by taking into account the variation of computational cost, symmetry, scalability, decomposability, etc. Meanwhile, the metrics of effectiveness, efficiency, or their hybrid, can be refined, as well as the significance test method in the distributed multiobjective environment.

5.4. *Real-world applications*

Because of its powerfulness, dEC can have and has seen a variety range of applications in science and engineering. Areas where dEAs have shown particular promise are problems with computationally expensive objective functions and extremely complex landscapes. The applications are so numerous and diverse that they exceed the scope of this paper. Hence we focus on several main fields and representative references here. Applications of dEAs in the literature can be classified into several categories, including the system design [99,137,74,118], resource scheduling [39,125,70,94], network planning [122,21,20], intelligent transportation [66,67,146], classifier optimization [129,95,6], feature extraction [77], and parameter training [37]. Compared to sequential EAs, the main benefits brought by dEAs are two-fold. On one hand, they improve the efficiency of EAs, and on the other, they enhance the global search ability and solution accuracy. In this sense, the dEC techniques improve the availability for solving real-world problems with large-scale, high-dimensional, and complex features.

6. Future directions

As surveyed in the above sections, significant efforts have been devoted to utilizing distributed computing resources to enhance the performance of EC. It is expected that dEC will continue to be a hot research topic because the complexity of real-world optimization problems is growing rapidly and there still exist many issues unexplored. In this section, we highlight several research directions of dEC.

6.1. *Highly scalable dEC*

Scalability is an important factor in distributed systems. For dEC, the increasing scale of real-world optimization problems requires the algorithm to scale up well to satisfy the intensive data processing need, but an overuse of computing resources is not cost-effective. Therefore, it is important to develop highly scalable dEC techniques that can increase or decrease resources, depending on the problem at hand. To address this issue, adopting a virtualization technique and an adaptive population size may be effective. Besides, some brand new branches of EC, such as the imperialist competitive algorithm (ICA) [60] and the social learning algorithm (SLA) [54], can be adopted as baseline algorithms for possible performance enhancement.

6.2. *Theoretical basis/proof of convergence*

As the communication bandwidth in a dEA is limited, it becomes harder to make clear how dEAs converge. Beside experimental analyses in the literature, studying the convergence of dEAs from a theoretical perspective will be appealing and meaningful. By building up a theoretical base for dEC, it may be convenient to develop some more powerful dEAs in the future.

6.3. *Systematical control of parameters*

Another issue introduced by the distributed paradigm is that dEAs have more parameters than sequential EAs. Compared with parameters of classical EAs, the newly introduced parameters have not yet been studied carefully, although they influence the performance of dEAs to a large extent. Therefore, it is crucial to control parameter settings systematically in dEAs or even automatically during the search process.

6.4. *Many-objective optimization*

Currently, MOEAs and dMOEAs have been effectively applied to deal with MOPs with a few, generally two or three, objectives. However, when facing the many-objective optimization problems involving four to tens of objectives, the performance of the algorithms deteriorates severely. The challenges arise from both the increased computational cost for evaluating the objective functions and the rapidly increased number of nondominated solutions in the population that breaks the Pareto selection pressure. Developing dEAs for many-objective optimization is promising since, by utilizing the distributed platform, it is now possible to manipulate a large population without incurring overlong computational time. The interplay of local evolution and global migration helps to seek a balance between convergence and diversity, which plays a decisive role in the performance of many-objective optimization. Moreover, the "many objectives, many processors (MOMP)" scheme, which optimizes a single objective on each processor and coordinates the optimization of different objectives during the intercommunication phase, forms an interesting and potentially useful future research direction.

6.5. *Evolutionary big data optimization*

The coming era of big data poses new challenges to data management and processing since the data involved are always large-scale, sparse, unstructured, uncertain, and spatial–temporal dependent. Owing to that EC does not require explicit mathematical models in problem solving, and that it can respond to application queries in a relatively short time, the EC paradigm can be considered as a promising solution in current data-driven optimization domain. Further, dEC, especially the cloud-based dEC and the mobile dEC being described in the next subsection, greatly improves the computational volume of EC and the cost-efficiency of deploying massive EC system, which is rather suitable for handling real-world big data optimization applications, such as information recommendation, disease prediction, and logistics transportation control, etc.

6.6. *Mobile evolutionary computation*

Smartphones possess useful computational capacity and the market is proliferating rapidly in recent years. The increasing quantity, mobile data connectivity and computational power have made smartphones a new and promising distributed system for dEC. Specifically, mobile crowdsourcing [19] is a probable form of deploying dEC on smartphones. Most existing mobile crowdsourcing applications emphasize on the sensing capability of smartphones, while in the context of dEC, the computation resource is of a central place. The challenge lies in that the deployment of dEAs on smartphones should not decrease the user experience, which is a basic requirement for the success of mobile crowdsourcing. Hence it is necessary to develop adaptive scheduling methods for executing mobile dEAs.

7. Conclusions

This article provides a comprehensive survey of the state-of-the-art distributed evolutionary algorithms and models. The models have been classified into two groups according to the task division mechanism. Population-distributed models include master-slave, island, cellular, hierarchical, and pool models, which parallelize an optimization task at population, individual, or operation levels. Dimension-distributed models include coevolution and multi-agent models that focus on the reduction of problem space. The characteristics of different models, such as the search behaviors, objectives, communication costs, scalability, and fault-tolerance, have been summarized and analyzed. It can be seen that these distributed models have different features and characteristics, which are suitable for developing different dEAs and solving different kinds of problems. We have also highlighted recent hotspots in dEC, including the cloud and MapReduce-based implementations, GPU and CUDA-based implementations, dMOEAs for multiobjective optimization, and real-world applications. Further, a number of future research directions have been discussed. Based on the survey, we believe that the study and development of distributed evolutionary computation will continue to be a vibrant and active field in the future.

References

[1] E. Alba, B. Dorronsoro, The exploration/exploitation tradeoff in dynamic cellular genetic algorithms, IEEE Trans. Evol. Comput. 9 (2) (2005) 126–142.
[2] E. Alba, B. Dorronsoro, M. Giacobini, M. Tomassini, Decentralized cellular evolutionary algorithms, Handb. Bioinspir. Algorithms Appl. 7 (2005) 103–120.
[3] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Trans. Evol. Comput. 6 (5) (2002) 443–462.
[4] E. Alba, J.M. Troya, A survey of parallel distributed genetic algorithms, Complexity 4 (4) (1999) 31–52.
[5] E. Alba, J.M. Troya, Cellular evolutionary algorithms: evaluating the influence of ratio, in: Parallel Problem Solving from Nature (PPSN), 2000, pp. 29–38.
[6] C. Anglano, M. Botta, NOW G-Net: learning classification programs on networks of workstations, IEEE Trans. Evol. Comput. 6 (5) (2002) 463–480.
[7] J. Apolloni, G. Leguizamón, J. García-Nieto, E. Alba, Island based distributed differential evolution: an experimental study on hybrid testbeds, in: Eighth International Conference on Hybrid Intelligent Systems, 2008, pp. 696–701.
[8] H. Bai, D. OuYang, X. Li, L. He, H. Yu, MAX-MIN ant system on GPU with CUDA, in: Fourth International Conference on Innovative Computing, Information and Control (ICICIC), 2009, pp. 801–804.
[9] A. Bollini, M. Piastra, Distributed and persistent evolutionary algorithms: a design pattern, in: Genetic Programming, 1999, pp. 173–183.
[10] P. Bouvry, F. Arbab, F. Seredynski, Distributed evolutionary optimization, in manifold: Rosenbrock's function case study, Inf. Sci. 122 (2) (2000) 141–159.
[11] J. Branke, H. Schmeck, K. Deb, S. Reddy, Parallelizing multi-objective evolutionary algorithms: cone separation, IEEE Congress on Evolutionary Computation (CEC) 2 (2004) 1952–1957.
[12] T. Burczynski, W. Kus, Optimization of structures using distributed and parallel evolutionary algorithms, in: Parallel Processing and Applied Mathematics, Springer, Berlin, Heidelberg, 2004, pp. 572–579.
[13] T. Burczyński, W. Kuś, A. Długosz, P. Orantek, Optimization and defect identification using distributed evolutionary algorithms, Eng. Appl. Artif. Intell. 17 (4) (2004) 337–344.
[14] S. Cahon, N. Melab, E.-G. Talbi, Building with paradisEO reusable parallel and distributed evolutionary algorithms, Parallel Comput. 30 (5) (2004) 677–697.
[15] S. Cahon, N. Melab, E.-G. Talbi, ParadisEO: a framework for the reusable design of parallel and distributed metaheuristics, J. Heurist. 10 (3) (2004) 357–380.
[16] E. Cantú-Paz, A survey of parallel genetic algorithms, Calcul. Paralleles Reseaux Syst. Repart. 10 (2) (1998) 141–171.
[17] E. Cantu-Paz, Efficient and Accurate Parallel Genetic Algorithms, vol.1, Springer, 2000.
[18] E. Cantu-Paz, Markov chain models of parallel genetic algorithms, IEEE Trans. Evol. Comput. 4 (3) (2000) 216–226.
[19] G. Chatzimilioudis, A. Konstantinidis, C. Laoudias, D. Zeinalipour-Yazti, Crowdsourcing with smartphones, IEEE Internet Comput. 16 (5) (2012) 36–44.
[20] Y. Chen, B. Wang, W.S. Lin, Y. Wu, K.R. Liu, Cooperative peer-to-peer streaming: an evolutionary game-theoretic approach, IEEE Trans. Circuits Syst. Video Technol. 20 (10) (2010) 1346–1357.
[21] J.-C. Creput, A. Koukam, T. Lissajoux, A. Caminada, Automatic mesh generation for mobile network dimensioning using evolutionary approach, IEEE Trans. Evol. Comput. 9 (1) (2005) 18–30.
[22] G. Danoy, P. Bouvry, O. Boissier, Dafo, a multi-agent framework for decomposable functions optimization, in: Knowledge-Based Intelligent Information and Engineering Systems, 2005, pp. 626–632.
[23] M. Davis, L. Liu, J.G. Elias, VLSI circuit synthesis using a parallel genetic algorithm, in: IEEE Congress on Evolutionary Computation (CEC), 1994, pp. 104–109.
[24] F. de Toro Negro, J. Ortega, E. Ros, S. Mota, B. Paechter, J. Martın, PSFGA: parallel processing and evolutionary computation for multiobjective optimisation, Parallel Comput. 30 (5) (2004) 721–739.
[25] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[26] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[27] J. Decraene, Y.Y. Cheng, M.Y.H. Low, S. Zhou, W. Cai, C.S. Choo, Evolving agent-based simulations in the clouds, in: Third International Workshop on Advanced Computational Intelligence, 2010, pp. 244–249.
[28] T. Desell, D.P. Anderson, M. Magdon-Ismail, H. Newberg, B.K. Szymanski, C.A. Varela, An analysis of massively distributed evolutionary algorithms, in: IEEE Congress on Evolutionary Computation (CEC), 2010, pp. 1–8.
[29] B. Dorronsoro, G. Danoy, A.J. Nebro, P. Bouvry, Achieving super-linear performance in parallel multi-objective evolutionary algorithms by means of cooperative coevolution, Comput. Oper. Res. 40 (6) (2013) 1552–1563.
[30] X. Du, L. Ding, L. Jia, Asynchronous distributed parallel gene expression programming based on estimation of distribution algorithm, in: International Conference on Natural Computation, 2008, pp. 433–437.
[31] M. Dubreuil, C. Gagné, M. Parizeau, Analysis of a master-slave architecture for distributed evolutionary computations, IEEE Trans. Syst. Man Cybern. B: Cybern. 36 (1) (2006) 229–235.
[32] J.J. Durillo, A.J. Nebro, jMetal: a java framework for multi-objective optimization, Adv. Eng. Softw. 42 (10) (2011) 760–771.
[33] J.J. Durillo, A.J. Nebro, E. Alba, The jmetal framework for multi-objective optimization: design and architecture, in: IEEE Congress on Evolutionary Computation (CEC), 2010, pp. 1–8.

[34] J.J. Durillo, A.J. Nebro, F. Luna, E. Alba, Solving three-objective optimization problems using a new hybrid cellular genetic algorithm, in: Parallel Problem Solving from Nature (PPSN), 2008, pp. 661–670.
[35] J.J. Durillo, A.J. Nebro, F. Luna, E. Alba, A study of master-slave approaches to parallelize NSGA-II, in: IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–8.
[36] J.J. Durillo, Q. Zhang, A.J. Nebro, E. Alba, Distribution of computational effort in parallel MOEA/D, in: Learning and Intelligent Optimization, Springer, 2011, pp. 488–502.
[37] M.G. Epitropakis, V.P. Plagianakos, M.N. Vrahatis, Hardware-friendly higher-order neural network training using distributed evolutionary algorithms, Appl. Soft Comput. 10 (2) (2010) 398–408.
[38] G. Escuela, Y. Cardinale, J. González, A Java-based distributed genetic algorithm framework, in: IEEE International Conference on Tools with Artificial Intelligence, 2007, pp. 437–441.
[39] G. Ewald, W. Kurek, M.A. Brdys, Grid implementation of a parallel multiobjective genetic algorithm for optimized allocation of chlorination stations in drinking water distribution systems: Chojnice case study, IEEE Trans. Syst. Man Cybern. C: Appl. Rev. 38 (4) (2008) 497–509.
[40] K.-L. Fok, T.-T. Wong, M.-L. Wong, Evolutionary computing on consumer-level graphics hardware, IEEE Intell. Syst. 22 (2) (2007) 69–78.
[41] G. Folino, C. Pizzuti, G. Spezzano, Training distributed GP ensemble with a selective algorithm based on clustering and pruning for pattern classification, IEEE Trans. Evol. Comput. 12 (4) (2008) 458–468.
[42] G. Folino, G. Spezzano, P-cage: an environment for evolutionary computation in peer-to-peer systems, in: Genetic Programming, 2006, pp. 341–350.
[43] C. Gagné, M. Parizeau, M. Dubreuil, Distributed beagle: An environment for parallel and distributed evolutionary computations, in: The 17th Annual International Symposium on High Performance Computing Systems and Applications, 2003, pp. 201–208.
[44] M. Garcia-Arenas, J.-J. Merelo, A.M. Mora, P. Castillo, G. Romero, J.L.J. Laredo, Assessing speed-ups in commodity cloud storage services for distributed evolutionary algorithms, in: IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 304–311.
[45] M. García-Arenas, J.J. Merelo Guervós, P. Castillo, J.L.J. Laredo, G. Romero, A.M. Mora, Using free cloud storage services for distributed evolutionary algorithms, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO), 2011, pp. 1603–1610.
[46] M. Giacobini, E. Alba, A. Tettamanzi, M. Tomassini, Modeling selection intensity for toroidal cellular evolutionary algorithms, in: Proceedings of the 6th Annual Conference on Genetic and Evolutionary Computation (GECCO), 2004, pp. 1138–1149.
[47] M. Giacobini, E. Alba, M. Tomassini, Selection intensity in asynchronous cellular evolutionary algorithms, in: Proceedings of the 5th Annual Conference on Genetic and Evolutionary Computation (GECCO), 2003, pp. 955–966.
[48] M. Giacobini, M. Preuss, M. Tomassini, Effects of scale-free and small-world topologies on binary coded self-adaptive CEA, in: Evolutionary Computation in Combinatorial Optimization, 2006, pp. 86–98.
[49] M. Giacobini, M. Tomassini, A. Tettamanzi, Modeling selection intensity for linear cellular evolutionary algorithms, in: Artificial Evolution, 2004, pp. 345–356.
[50] M. Giacobini, M. Tomassini, A. Tettamanzi, Takeover time curves in random and small-world structured populations, in: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO), 2005, p. 1333-L1340.
[51] M. Giacobini, M. Tomassini, A.G. Tettamanzi, E. Alba, Selection intensity in cellular evolutionary algorithms for regular lattices, IEEE Trans. Evol. Comput. 9 (5) (2005) 489–505.
[52] C.-K. Goh, K. Chen Tan, A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization, IEEE Trans. Evol. Comput. 13 (1) (2009) 103–127.
[53] D.E. Goldberg, K. Deb, A comparative analysis of selection schemes used in genetic algorithms, Urbana 51 (1991) 61801–62996.
[54] Y.-J. Gong, J. Zhang, Y. Li, From the social learning theory to a social learning algorithm for global optimization, in: IEEE International Conference on Systems, Man and Cybernetics (SMC), 2014, pp. 222–227.
[55] D.L. Gonzalez, F.F. de Vega, On the intrinsic fault-tolerance nature of Parallel Genetic Programming, in: EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing, 2007, pp. 450–458.
[56] F. Herrera, M. Lozano, Gradual distributed real-coded genetic algorithms, IEEE Trans. Evol. Comput. 4 (1) (2000) 43–63.
[57] F. Herrera, M. Lozano, C. Moraga, Hierarchical distributed genetic algorithms, Int. J. Intell. Syst. 14 (11) (1999) 1099–1121.
[58] J.I. Hidalgo, F. Fernández, Balancing the computation effort in genetic algorithms, in: IEEE Congress on Evolutionary Computation (CEC), 2005, pp. 1645–1652.
[59] J.I. Hidalgo, J. Lanchares, F. Fernández de Vega, D. Lombraña, Is the island model fault tolerant? in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO), 2007, pp. 2737–2744.
[60] S. Hosseini, A. Al Khaled, A survey on the imperialist competitive algorithm metaheuristic: implementation in engineering domain and directions for future research Appl. Soft Comput. 24 (2014) 1078–1094.
[61] I. Iimura, K. Hamaguchi, T. Ito, S. Nakayama, A study of distributed parallel processing for queen ant strategy in ant colony optimization, in: International Conference on Parallel and Distributed Computing, Applications and Technologies, 2005, pp. 553–557.
[62] T. Ishimizu, K. Tagawa, A structured differential evolutions for various network topologies, Int. J. Comput. Commun. 4 (1) (2010) 1–8.
[63] M.A. Ismail, Parallel genetic algorithms (PGAs): master-slave paradigm approach using MPI, in: E-Tech, 2004, pp. 83–87.
[64] C. Jin, C. Vecchiola, R. Buyya, MRPGA: an extension of mapreduce for parallelizing genetic algorithms, in: IEEE Fourth International Conference on eScience, 2008, pp. 214–221.
[65] F.M. Johar, F.A. Azmin, M.K. Suaidi, A.S. Shibghatullah, B.H. Ahmad, S.N. Salleh, M.Z.A.A. Aziz, M. Md Shukor, A review of genetic algorithms and parallel genetic algorithms on graphics processing unit (GPU), in: 2013 IEEE International Conference on Control System, Computing and Engineering (ICCSCE), 2013, pp. 264–269.
[66] L. Kattan, B. Abdulhai, Distributed evolutionary estimation of dynamic traffic origin/destination, in: 13th International IEEE Conference on Intelligent Transportation Systems, 2010, pp. 911–916.
[67] L. Kattan, B. Abdulhai, Sensitivity analysis of an evolutionary-based time-dependent origin/destination estimation framework, IEEE Trans. Intell. Transp. Syst. 13 (3) (2012) 1442–1453.
[68] M. Kirley, R. Stewart, An analysis of the effects of population structure on scalable multiobjective optimization problems, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO), 2007, pp. 845–852.
[69] M. Kirley, R. Stewart, Multiobjective evolutionary algorithms on complex networks, in: Evolutionary Multi-criterion Optimization, 2007, pp. 81–95.
[70] Y.-K. Kwok, I. Ahmad, Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm, J. Parallel Distrib. Comput. 47 (1) (1997) 58–77.
[71] J.L.J. Laredo, P.A. Castillo, A.M. Mora, J. Merelo, Evolvable agents, a fine grained approach for distributed evolutionary computing: walking towards the peer-to-peer computing frontiers, Soft Comput. 12 (12) (2008) 1145–1156.
[72] X. Li, X. Yao, Cooperatively coevolving particle swarms for large scale optimization, IEEE Trans. Evol. Comput. 16 (2) (2012) 210–224.
[73] C. Liang, C. Chung, K. Wong, X. Duan, Parallel optimal reactive power flow based on cooperative co-evolutionary differential evolution and power system decomposition, IEEE Trans. Power Syst. 22 (1) (2007) 249–257.
[74] J. Lienig, A parallel genetic algorithm for performance-driven VLSI routing, IEEE Trans. Evol. Comput. 1 (1) (1997) 29–39.
[75] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, B.-S. Lee, Efficient hierarchical parallel genetic algorithms using grid computing, Future Gener. Comput. Syst. 23 (4) (2007) 658–670.
[76] D.H. Lim, H.N. Luong, C.W. Ahn, A novel differential evolution incorporated with parallel processing mechanism, in: 2nd International Workshop on Intelligent Systems and Applications, 2010, pp. 1–4.
[77] J. Liu, Y.Y. Tang, Y. Cao, An evolutionary autonomous agents approach to image feature extraction, IEEE Trans. Evol. Comput. 1 (2) (1997) 141–158.
[78] X. Llora, A. Verma, R.H. Campbell, D.E. Goldberg, When huge is routine: scaling genetic algorithms and estimation of distribution algorithms via data-intensive computing, in: Parallel and Distributed Computational Intelligence, Springer, Berlin, Heidelberg, 2010, pp. 11–41.
[79] Y. Lorion, T. Bogon, I.J. Timm, O. Drobnik, An agent based parallel particle swarm optimization-APPSO, in: IEEE Swarm Intelligence Symposium, 2009, pp. 52–59.
[80] M. Manfrin, M. Birattari, T. Stützle, M. Dorigo, Parallel ant colony optimization for the traveling salesman problem, in: Ant Colony Optimization and Swarm Intelligence, 2006, pp. 224–234.
[81] A.W. McNabb, C.K. Monson, K.D. Seppi, Parallel PSO using mapreduce, in: IEEE Congress on Evolutionary Computation (CEC), 2007, pp. 7–14.
[82] N. Melab, E.-G. Talbi, GPU-based island model for evolutionary algorithms, in: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO), 2010, pp. 1089–1096.
[83] N. Melab, E.-G. Talbi, Parallel hybrid evolutionary algorithms on GPU, in: IEEE Congress on Evolutionary Computation (CEC), 2010, pp. 1–8.
[84] A. Mendiburu, J.A. Lozano, J. Miguel-Alonso, Parallel implementation of EDAs based on probabilistic graphical models, IEEE Trans. Evol. Comput. 9 (4) (2005) 406–423.
[85] J. Merelo, A.M. Mora, C.M. Fernandes, A.I. Esparcia-Alcázar, Designing and testing a pool-based evolutionary algorithm, Nat. Comput. 12 (2) (2013) 149–162.
[86] J.-J. Merelo-Guervós, A. Mora, J.A. Cruz, A.I. Esparcia-Alcazar, Pool-based distributed evolutionary algorithms using an object database, in: Applications of Evolutionary Computation, Springer, Berlin, Heidelberg, 2012, pp. 446–455.
[87] J.J. Merelo-Guervos, A. Mora, J.A. Cruz, A.I. Esparcia-Alcazar, C. Cotta, Scaling in distributed evolutionary algorithms with persistent population, in: IEEE Congress on Evolutionary Computation (CEC), 2012, pp. 1–8.
[88] J.J. Merelo-Guervós, A.M. Mora, C.M. Fernandes, A.I. Esparcia-Alcazar, J.L.J. Laredo, Pool vs. island based evolutionary algorithms: an initial exploration, in: International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2012, pp. 19–24.
[89] R. Michel, M. Middendorf, An island model based ant system with lookahead for the shortest supersequence problem, in: Parallel Problem Solving from Nature (PPSN), 1998, pp. 692–701.

[90] S. Mostaghim, J. Branke, H. Schmeck, Multi-objective particle swarm optimization on computer grids, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (PPSN), 2007, pp. 869–875.
[91] L. Mussi, Y.S. Nashed, S. Cagnoni, GPU-based asynchronous particle swarm optimization, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO), 2011, pp. 1555–1562.
[92] A.J. Nebro, J.J. Durillo, A study of the parallelization of the multi-objective metaheuristic MOEA/D, in: Learning and Intelligent Optimization, Springer, Berlin, Heidelberg, 2010, pp. 303–317.
[93] A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, E. Alba, Mocell: a cellular genetic algorithm for multiobjective optimization, Int. J. Intell. Syst. 24 (7) (2009) 726–746.
[94] S. Nesmachnow, H. Cancela, E. Alba, A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling, Appl. Soft Comput. 12 (2) (2012) 626–639.
[95] Y. Nojima, S. Mihara, H. Ishibuchi, Ensemble classifier design by parallel distributed implementation of genetic fuzzy rule selection for large data sets, in: IEEE Congress on Evolutionary Computation (CEC), 2010, pp. 1–8.
[96] P.C. Ordeshook, Game Theory and Political Theory: An Introduction, Cambridge University Press, 1986.
[97] M. Pedemonte, S. Nesmachnow, H. Cancela, A survey on parallel ant colony optimization, Appl. Soft Comput. 11 (8) (2011) 5181–5197.
[98] P.C. Pendharkar, A multi-agent memetic algorithm approach for distributed object allocation, J. Comput. Sci. 2 (4) (2011) 353–364.
[99] H. Pierreval, J.-L. Paris, Distributed evolutionary algorithms for simulation optimization, IEEE Trans. Syst. Man Cybern. A: Syst. Hum. 30 (1) (2000) 15–24.
[100] D.A.L. Piriyakumar, P. Levi, A new approach to exploiting parallelism in ant colony optimization, in: International Symposium on Micromechatronics and Human Science, 2002, pp. 237–243.
[101] M.A. Potter, K.A. De Jong, Cooperative coevolution: An architecture for evolving coadapted subcomponents, Evol. Comput. 8 (1) (2000) 1–29.
[102] T. Ray, X. Yao, A cooperative coevolutionary algorithm with correlation based adaptive variable partitioning, in: IEEE Congress on Evolutionary Computation (CEC), 2009, pp. 983–989.
[103] D. Robilliard, V. Marion, C. Fonlupt, High performance genetic programming on GPU, in: Proceedings of Bio-Inspired Algorithms for Distributed Systems, 2009, pp. 85–94.
[104] G. Roy, H. Lee, J.L. Welch, Y. Zhao, V. Pandey, D. Thurston, A distributed pool architecture for genetic algorithms, in: IEEE Congress on Evolutionary Computation (CEC), 2009, pp. 1177–1184.
[105] A. Ruiz-Andino, L. Araujo, F. Sáenz, J.J. Ruz, A hybrid evolutionary approach for solving constrained optimization problems over finite domains, IEEE Trans. Evol. Comput. 4 (4) (2000) 353–372.
[106] S.M. Said, M. Nakamura, Asynchronous strategy of parallel hybrid approach of GA and EDA for function optimization, in: International Conference on Networking and Computing, 2012, pp. 420–428.
[107] S.M. Said, M. Nakamura, Parallel enhanced hybrid evolutionary algorithm for continuous function optimization, in: International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2012, pp. 125–131.
[108] J. Sarma, K. De Jong, An analysis of the effects of neighborhood size and shape on local selection algorithms, in: Parallel Problem Solving From Nature (PPSN), 1996, pp. 236–244.
[109] B. Schönfisch, A. de Roos, Synchronous and asynchronous updating in cellular automata, BioSystems 51 (3) (1999) 123–143.
[110] J.F. Schutte, J.A. Reinbolt, B.J. Fregly, R.T. Haftka, A.D. George, Parallel global optimization with the particle swarm algorithm, Int. J. Numer. Methods Eng. 61 (13) (2004) 2296–2315.
[111] I. Scriven, A. Lewis, D. Ireland, J. Lu, Decentralised distributed multiple objective particle swarm optimisation using peer to peer networks, in: IEEE Congress on Evolutionary Computation (CEC), 2008, pp. 2925–2928.
[112] M. Sefrioui, J. Périaux, A hierarchical genetic algorithm using multiple models for optimization, in: Parallel Problem Solving from Nature (PPSN), 2000, pp. 879–888.
[113] F. Seredynski, Loosely coupled distributed genetic algorithms, in: Parallel Problem Solving from Nature (PPSN), 1994, pp. 514–523.
[114] F. Seredynski, Competitive coevolutionary multi-agent systems: The application to mapping and scheduling problems, J. Parallel Distrib. Comput. 47 (1) (1997) 39–57.
[115] F. Seredynski, A.Y. Zomaya, P. Bouvry, Function optimization with coevolutionary algorithms, in: Intell. Inf. Process. Web Min., 2003, pp. 13–22.
[116] D. Sherry, K. Veeramachaneni, J. McDermott, U.-M. OReilly, Flex-GP: genetic programming on the cloud, in: Applications of Evolutionary Computation, Springer, Berlin, Heidelberg, 2012, pp. 477–486.
[117] N. Soca, J.L. Blengio, M. Pedemonte, P. Ezzatti, PUGACE, a cellular evolutionary algorithm framework on GPUs, in: IEEE Congress on Evolutionary Computation (CEC), 2010, pp. 1–8.
[118] J. Starzynski, R. Szmurlo, J. Kijanowski, B. Dawidowicz, B. Sawicki, S. Wincenciak, Distributed evolutionary algorithm for optimization in electromagnetics, IEEE Trans. Magn. 42 (4) (2006) 1243–1246.
[119] T. Stützle, Parallelization strategies for ant colony optimization, in: Parallel Problem Solving from Nature (PPSN), 1998, pp. 722–731.
[120] S. Su, C. Chung, K. Wong, Y. Fung, D. Yeung, Fault tolerant differential evolution based optimal reactive power flow, in: International Conference on Machine Learning and Cybernetics, 2006, pp. 4083–4088.
[121] R. Subbu, A.C. Sanderson, Modeling and convergence analysis of distributed coevolutionary algorithms, IEEE Trans. Syst. Man Cybern. B: Cybern. 34 (2) (2004) 806–822.
[122] R. Subbu, A.C. Sanderson, Network-based distributed planning using coevolutionary agents: architecture and evaluation, IEEE Trans. Syst. Man Cybern. A: Syst. Hum. 34 (2) (2004) 257–269.
[123] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-parameter Optimization, KanGAL Report 2005005, 2005.
[124] K. Tagawa, T. Ishimizu, Concurrent differential evolution based on MapReduce, Int. J. Comput. 4 (4) (2010) 161–168.
[125] K.C. Tan, E.F. Khor, J. Cai, C. Heng, T.H. Lee, Automating the drug scheduling of cancer chemotherapy via evolutionary computation, Artif. Intell. Med. 25 (2) (2002) 169–185.
[126] K.C. Tan, W. Peng, T.H. Lee, J. Cai, Development of a distributed evolutionary computing package, in: IEEE Congress on Evolutionary Computation (CEC), 2003, pp. 77–84.
[127] K.C. Tan, A. Tay, J. Cai, Design and implementation of a distributed evolutionary computing software, IEEE Trans. Syst. Man Cybern. C: Appl. Rev. 33 (3) (2003) 325–338.
[128] K.C. Tan, Y. Yang, C.K. Goh, A distributed cooperative coevolutionary algorithm for multiobjective optimization, IEEE Trans. Evol. Comput. 10 (5) (2006) 527–549.
[129] K.C. Tan, Q. Yu, T.H. Lee, A distributed evolutionary classifier for knowledge discovery in data mining, IEEE Trans. Syst. Man Cybern. C: Appl. Rev. 35 (2) (2005) 131–142.
[130] D.K. Tasoulis, N.G. Pavlidis, V.P. Plagianakos, M.N. Vrahatis, Parallel differential evolution, in: IEEE Congress on Evolutionary Computation (CEC), 2004, pp. 2023–2029.
[131] M. Tomassini, Parallel and distributed evolutionary algorithms: a review, Citeseer (1999).
[132] M. Tomassini, Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time, Springer, 2005.
[133] A. Umbarkar, M. Joshi, Review of parallel genetic algorithm based on computing paradigm and diversity in search space, ICTACT J. Soft Comput. 3 (2013) 615–622.
[134] D.A. Van Veldhuizen, J.B. Zydallis, G.B. Lamont, Considerations in engineering parallel multiobjective evolutionary algorithms, IEEE Trans. Evol. Comput. 7 (2) (2003) 144–173.
[135] A. Verma, X. Llora, D.E. Goldberg, R.H. Campbell, Scaling genetic algorithms using mapreduce, in: Ninth International Conference on Intelligent Systems Design and Applications, 2009, pp. 13–18.
[136] L.d.P. Veronese, R.A. Krohling, Differential evolution algorithm on the GPU with C-CUDA, in: IEEE Congress on Evolutionary Computation (CEC), 2010, pp. 1–7.
[137] J.G. Vlachogiannis, K.Y. Lee, Determining generator contributions to transmission system using parallel vector evaluated particle swarm optimization, IEEE Trans. Power Syst. 20 (4) (2005) 1765–1774.
[138] M. Weber, F. Neri, V. Tirronen, Distributed differential evolution with explorative-exploitative population families, Genet. Program. Evol. Mach. 10 (4) (2009) 343–371.
[139] M. Weber, V. Tirronen, F. Neri, Scale factor inheritance mechanism in distributed differential evolution, Soft Comput. 14 (11) (2010) 1187– 1207.
[140] D. Whitley, T. Starkweather, Genitor II: A distributed genetic algorithm, J. Exp. Theor. Artif. Intell. 2 (3) (1990) 189–214.
[141] W. Wickramasinghe, M. van Steen, A. Eiben, Peer-to-peer evolutionary algorithms with adaptive autonomous selection, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO), 2007, pp. 1460–1467.
[142] B. Wu, G. Wu, M. Yang, A mapreduce based ant colony optimization approach to combinatorial optimization problems, in: International Conference on Natural Computation (ICNC), 2012, pp. 728–732.
[143] S. Xiong, F. Li, Parallel strength pareto multi-objective evolutionary algorithm for optimization problems, in: IEEE Congress on Evolutionary Computation (CEC), 2003, pp. 2712–2718.
[144] L. Xu, F. Zhang, Parallel particle swarm optimization for attribute reduction, in: ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, vol. 1, 2007, pp. 770–775.
[145] Z. Yang, K. Tang, X. Yao, Large scale evolutionary optimization using cooperative coevolution, Inf. Sci. 178 (15) (2008) 2985–2999.
[146] B. Yu, Z. Yang, X. Sun, B. Yao, Q. Zeng, E. Jeppesen, Parallel genetic algorithm in bus route headway optimization, Appl. Soft Comput. 11 (8) (2011) 5081–5091.
[147] W. Yu, W. Zhang, Study on function optimization based on master-slave structure genetic algorithm, in: International Conference on Signal Processing, vol. 3, 2006, pp. 1–4.
[148] C. Zhang, J. Chen, B. Xin, Distributed memetic differential evolution with the synergy of Lamarckian and Baldwinian learning, Appl. Soft Comput. 13 (5) (2013) 2947–2959.
[149] J. Zhang, H.S.-H. Chung, W.-L. Lo, Pseudocoevolutionary genetic algorithms for power electronic circuits optimization, IEEE Trans. Syst. Man Cybern. C: Appl. Rev. 36 (4) (2006) 590–598.
[150] Q. Zhang, H. Li, MOEA/D: A multiobjective evolutionary algorithm based on decomposition, IEEE Trans. Evol. Comput. 11 (6) (2007) 712–731.

[151] J. Zhao, W. Wang, W. Pedrycz, X. Tian, Online parameter optimization-based prediction for converter gas system by parallel strategies, IEEE Trans. Control Syst. Technol. 20 (3) (2012) 835–845.
[152] W. Zhao, S. Alam, H.A. Abbass, MOCCA-II: A multi-objective co-operative co-evolutionary algorithm, Appl. Soft Comput. 23 (2014) 407–416.
[153] C. Zhou, Fast parallelization of differential evolution algorithm using MapReduce, in: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO), 2010, pp. 1113–1114.
[154] Y. Zhou, Y. Tan, GPU-based parallel particle swarm optimization, in: IEEE Congress on Evolutionary Computation (CEC), 2009, pp. 1493–1500.
[155] W. Zhu, Nonlinear optimization with a massively parallel evolution strategy-pattern search algorithm on graphics hardware, Appl. Soft Comput. 11 (2) (2011) 1770–1781.