Parallelism and Evolutionary Algorithms

Enrique Alba and Marco Tomassini

***Abstract*—This paper contains a modern vision of the parallelization techniques used for evolutionary algorithms (EAs). The work is motivated by two fundamental facts: first, the different families of EAs have naturally converged in the last decade while parallel EAs (PEAs) seem still to lack unified studies, and second, there is a large number of improvements in these algorithms and in their parallelization that raise the need for a comprehensive survey. We stress the differences between the EA model and its parallel implementation throughout the paper. We discuss the advantages and drawbacks of PEAs. Also, successful applications are mentioned and open problems are identified. We propose potential solutions to these problems and classify the different ways in which recent results in theory and practice are helping to solve them. Finally, we provide a highly structured background relating PEAs in order to make researchers aware of the benefits of decentralizing and parallelizing an EA.**

***Index Terms*—Evolutionary algorithms, first hitting time, population, time complexity.**

I. Introduction

EVOLUTIONARY algorithms (EAs) are stochastic search methods that have been applied successfully in many search, optimization, and machine learning problems [68], [49], [88], [14]. Unlike most other optimization techniques, EAs maintain a population of μ-encoded tentative solutions that are manipulated competitively by applying some variation operators to find a satisfactory, if not globally optimum solution. Several other heuristics such as simulated annealing [74], tabu search [48], and their combinations and variations [123] have been used with comparable results but will not be reviewed here.

The goal of this paper is to bring uniformity and structure to the different research issues concerning parallel EAs (PEAs). We will address the algorithms, as well as the machines and software for PEAs, and at the same time, we will stress their relationship with practice, theory, and applications in a novel way. Thus, we need to offer some history, as well as the present and future trends in this field of the evolutionary computation (EC) discipline. Our review is an up-to-date discussion about the main parallel achievements in the algorithmic families included in the EC. We review parallel models, parallel implementations, theoretical issues, and the applications of these algorithms, stressing the importance of unification in the PEA's field.

Let us begin by outlining the skeleton of a standard EA. An EA (see the following pseudocode) proceeds in an iterative manner by generating new populations $P(t)$ of individuals from the old ones $(t = 0, t = 1, t = 2, \ldots)$. Every individual in the population is the encoded (binary, real, ...) version of a tentative solution. An evaluation function associates a fitness value to every individual indicating its suitability to the problem. The canonical algorithm applies stochastic operators such as selection, crossover, and mutation on an initially random population in order to compute a whole generation of new individuals. In a general formulation, we apply variation operators to create a temporary population $P'(t)$, evaluate the resulting individuals, and get a new population $P(t+1)$ by either using $P'(t)$ or, optionally, $P(t)$. The halting condition is usually set as reaching a preprogrammed number of iterations of the algorithm, or to find an individual with a given error if the optimum, or an approximation to it, is known beforehand.

```
Evolutionary Algorithm
t := 0;
initialize and evaluate [P(t)];
while not stop_condition do
  P'(t) := variation [P(t)];
  evaluate [P'(t)];
  P(t+1) := select [P'(t),P(t)];
  t := t + 1;
end while;
```

Often, the fields of evolutionary computing, neural networks, and fuzzy logic, are listed together as techniques for solving problems by using numeric knowledge representation, in opposition to traditional artificial intelligence, where symbolic knowledge representation is used. Unlike conventional algorithms, they are tolerant of imprecision, uncertainty, and partial truth. These features make them less brittle than standard approaches and, as a consequence, they offer adaptivity. This broader research field, which also comprises other techniques such as rough sets and probabilistic networks, is known as *soft computing* (see [127] for further readings on this matter).

The right side of Fig. 1 details the well-accepted subclasses of EAs, namely genetic algorithms (GA), evolutionary programming (EP), evolution strategies (ES), and others not shown here.[1] (See [13] for learning how similar the different EA families are.)

For nontrivial problems, executing the reproductive cycle of a simple EA on long individuals and/or large populations requires high computational resources. In fact, although many EA families, such as GAs or ESs, use a string of numeric values, many other EAs (e.g., genetic programming (GP) and EP) use individuals having a *complex internal data structure* representing

Manuscript received January 29, 2001; revised September 27, 2001 and February 23, 2002. The work of E. Alba was supported in part by the Spanish National CICYT Project TIC1999-0754-C03-03.

E. Alba is with the Department of Computer Science, University of Málaga, Málaga, Spain (e-mail: eat@lcc.uma.es).

M. Tomassini is with the Institute of Computer Science, University of Lausanne, CH-1015 Lausanne, Switzerland.

Digital Object Identifier 10.1109/TEVC.2002.800880.

[1]The reader can find a great deal of structured and useful information about evolutionary computing in the *Handbook of EC* [14].

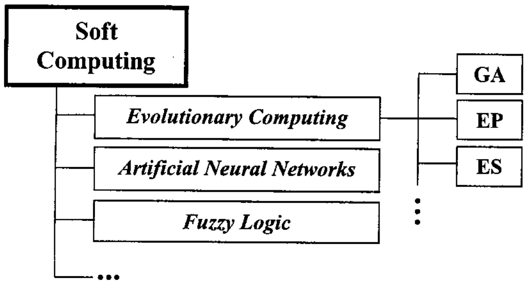


Fig. 1. Soft computing and EAs.

the problem in a more sophisticated way (e.g., a symbol tree). In general, evaluating a fitness function for every individual in order to compute its quality as a solution is frequently the most costly operation of the EA. Consequently, a variety of algorithmic issues are being studied to design efficient EAs. These issues usually consist of defining new operators, hybrid algorithms, parallel models, and so on. We develop our work in one of such research lines consisting in using parallel models of EAs. In this field, there exists a large number of implementations and algorithms, although not much effort has been devoted to the algorithm design. We can find in [118] a first attempt at designing algorithms in the framework of the PAPAGENA European project; also, in [9], a software engineering point of view for PEAs can be found.

But efficiency and design are not the only important issues with PEAs [64]. First of all, PEAs are naturally prone to parallelism, since most variation operations can be easily undertaken in parallel. Using a PEA often leads to not only a faster algorithm, but also to a superior numerical performance. However, the truly interesting observation is that the use of a *structured population*, that is, a spatial distribution of individuals, either in the form of a set of islands [126] or a diffusion grid [81], [116], is the responsible of such benefits. As a consequence, many authors do not use a parallel machine at all to run structured-population models, and still get better results than with serial traditional EAs [54].

In classical PEA studies (e.g., in PGAs [26]), it is assumed that the model maps directly onto the parallel hardware, thus making no distinction between the model and its implementation. However, once a structured-population model has been designed, it can be implemented in any monoprocessor or parallel machine. This conception (see Section IV) of model versus implementation raises several questions. First, any EA can be run in parallel, although high efficiency is not always possible [17]. Second, analyzing PEAs suggests the necessity of using a complex and heterogeneous test suite [133] (e.g., multimodal, deceptive, and epistatic problems). Third, special care must be taken to ensure that the experiments will be replicable to allow future extensions. Finally, some questions are open in relation to the physical, numerical, and parallel execution of the models. We must notice that additional parameters are needed to determine the search in a PEA, thus requiring further research to understand their importance.

PEAs have deserved several reviews in the past that can be used for tracking their evolution in time. (See, for example, [26], [1], [5], [21].) We now turn to justify the *contributions* and the distinctive aspects of the present work with respect to previous similar review work. We think that this review represents an original way of presenting the material that is scattered in many places to practitioners of the EA field. It carefully separates models from implementations, and accurately describes suitable computing environments and communication paradigms taking into account modern trends in distributed systems. This, to our knowledge, has never been done before in a single place. Moreover, theoretical aspects are described in a readable way and with many key references, something that has also been overlooked in the past, since theory is quite new and empirical considerations were preferred in previous reviews. To this, we add a very broad spectrum of related information that includes a section on nonuniform PEAs, the seldom discussed concept of speedup, and an effort to include EAs other than GAs. All this has not been considered in existing previous work.

The organization of this paper is as follows. First, we offer a summary of the most important milestones in the PEA history. Then, *panmictic*, i.e., the standard model in which the whole population is dealt with as a single pool of individuals, and structured EAs (where some partition of the single pool is undertaken) are discussed from a unified viewpoint. Next, the implementation of EAs on parallel machines is presented by discussing hardware, software, and algorithmic alternatives for building an efficient PEA. A further section is devoted to the theoretical advances relating structured and PEAs, including a discussion of speedup in PEAs. Finally, several PEA classifications are given to offer a full overview of the state of the art. Future trends and open problems are identified throughout, and a carefully chosen bibliography is referenced to guide the reader.

II. History

The intrinsically parallel and distributed nature of EAs did not escape the attention of early researchers. Holland [68] offered some steps toward defining a parallel computing architecture for reproductive plans. In fact, the first ideas about using multiple competing subpopulations can be traced back to the work of Bossert [19] who proposed them as a procedure to improve diversity and delay stagnation. However, although the main ideas were understood, the technology of parallel and distributed computing was in a primitive stage in the 1960s. It was, therefore, difficult to create practical implementations and simulations. The field had to wait until the early 1980s for suitable parallel implementations to appear. Grefenstette [59] was one of the first in examining a number of issues pertaining to the parallel implementations of GAs in 1981. Grosso [61] is another early attempt to introduce parallelism through the use of a spatial multipopulation model. This was followed by more systematic studies by Cohoon, Tanese, Pettey and Leuze, Gorges-Schleuter and Mühlenbein, and Manderick and Spiessens. Tanese [125] and Cohoon [30] employed the novel (at the time) hypercube parallel architecture by placing subpopulations at the hypercube nodes. For ES, Rudolph [105] implemented one of the first distributed models. For EP, Duncan [38] was an important milestone. The early work of Pettey and Leuze [97] is also significant because it was the first in trying to model multipopulation GA dynamics theoretically.

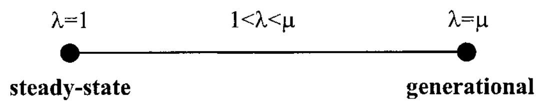


Fig. 2. Panmictic EAs: from steady state to generational algorithms.

All these studies pertain to what has been called a coarse-grain or island model (see Section IV). Another related spatial model was popularized by early work of Gorges–Schleuter [55]. It is called a diffusion, cellular, or fine-grained model (see Section IV) and is based on a spatially distributed population in which genetic interactions may only take place in a small neighborhood of each individual. Related work was done at the same time by Manderick and Spiessen [81].

III. Panmictic Evolutionary Algorithms

In EC, it is customary to find algorithm families in which the population structure is panmictic (see Section I). Thus, selection takes place globally and any individual can potentially mate with any other. The same holds for the replacement operator, where any individual can potentially leave the pool and be replaced by a new one. A different (decentralized) selection model exist in which individuals are arranged spatially, therefore giving place to *structured EAs* (see Section IV). Most other operators, such as recombination or mutation, can be readily applied to these two models.

There exist two popular classes of panmictic EAs having different granularity at the reproductive step [122]. The first one is called a "generational" model, in which a whole new population of λ individuals replaces the old one (right part of Fig. 2, where μ is the population size). The second type is called "steady state," since usually one or two new individuals are created at every step of the algorithm and then they are inserted back into the population, consequently coexisting with their parents. In the mean region, there exists a plethora of selection models generically termed as "generation gap" algorithms, in which a given percentage of the individuals are replaced with the new ones. Clearly, generational and steady-state selection are two special subclasses of generation gap algorithms.

Centralized versions of selection are typically found in serial EAs, although some parallel implementations have also used it. For example, the *global parallelism* approach evaluates in parallel the individuals of the population while still using a centralized selection performed sequentially in the main processor guiding the base algorithm [76]. This algorithm is the same as the sequential one, although it is faster, especially for time-consuming objective functions. Usually, the other parts of the algorithm are not worth parallelizing, unless some population structuring principle is used (see Section IV).

Most PEAs found in the literature utilize some kind of spatial disposition for the individuals, and then parallelize the resulting chunks in a pool of processors. We must stress at this point of the discussion that parallelization is achieved by first structuring the panmictic algorithm and then parallelizing it. This is why we distinguish throughout the paper between structuring populations and making parallel implementations, since the same structured EA can admit many different implementations. In fact, we have just drawn some ideas as to how panmictic EAs can be parallelized. Section IV is devoted to explaining different ways of structuring the populations. The resulting model can be executed in parallel or not, although some structured models suggest a straightforward parallel implementation.

IV. Structured EAs

There exists a long tradition in using structured populations in EC, especially associated to parallel implementations. Among the most widely known types of structured EAs, the *distributed* (dEA) and *cellular* (cEA) algorithms are very popular optimization procedures [5].

Decentralizing a single population can be achieved by partitioning it into several subpopulations, where island EAs are run performing sparse exchanges of individuals (dEAs), or in the form of neighborhoods (cEAs). These two EA types, along with a panmictic EA, are schematically depicted in Fig. 3.

In dEAs, additional parameters controlling when migration occurs and how migrants are selected/incorporated from/to the source/target islands are needed [17], [126]. In cEAs, the existence of overlapped small neighborhoods helps in exploring the search space [16]. These two kinds of EAs seem to provide a better sampling of the search space and improve the numerical and runtime behavior of the basic algorithm in many cases [10], [54].

The main difference in a cEA, with respect to a panmictic EA, is its decentralized selection and variation. In cEAs, the reproductive loop is performed inside every one of the numerous individual pools. In a cEA, one given individual has its own pool of potential mates defined by neighboring individuals; at the same time, one individual belongs to many pools. This one-dimensional (1-D) or two-dimensional (2-D) structure with overlapped neighborhoods is used to provide a smooth diffusion of good solutions across the *grid*. We mentioned a 2-D grid of individuals due to its generality [108]. A cEA can be implemented in a distributed memory MIMD computer [84], although its more natural implementation is on a SIMD computer (see some well-accepted computer architecture descriptions in Section V-A and also Fig. 5).

A dEA is a multipopulation (island) model performing sparse exchanges of individuals among the elementary populations. This model can be readily implemented in distributed memory MIMD computers, which provides one main reason for its popularity. A migration policy controls the kind of dEA being used. The migration policy must define the island topology, when migration occurs, which individuals are being exchanged, the synchronization among the subpopulations, and the kind of integration of exchanged individuals within the target subpopulations. The advantages of a distributed model (either running on separate processors or not) is that it is usually faster than a panmictic EA. The reason for this is that the run time and the number of evaluations are potentially reduced thanks to its separate search in several regions from the problem space. A high diversity and species formation are two of their well-reported features.

In Fig. 4, we plot a three-dimensional (3-D) representation of structured algorithms based on the number of subpopulations, the number of individuals in each one, and the degree of interaction among them.

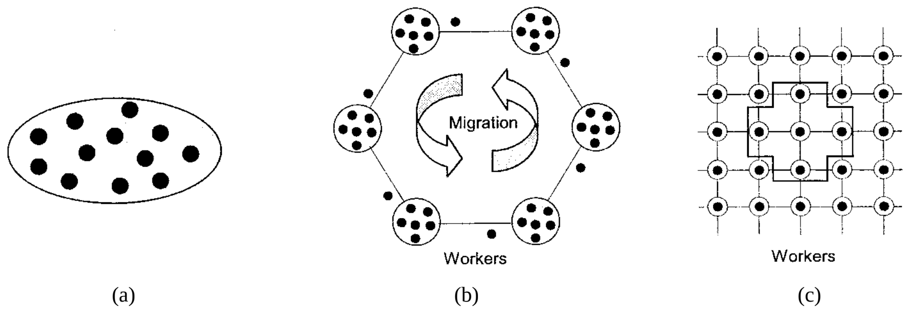


Fig. 3. A panmictic EA has all its individual's black points in the same population and, thus, (a) each of them can potentially mate with any other. Structuring the population usually leads to a distinction between (b) dEAs and (c) cEAs. These latter models are usually implemented on MIMD and SIMD machines (see Fig. 5), respectively, although nothing prevents other kinds of implementations.

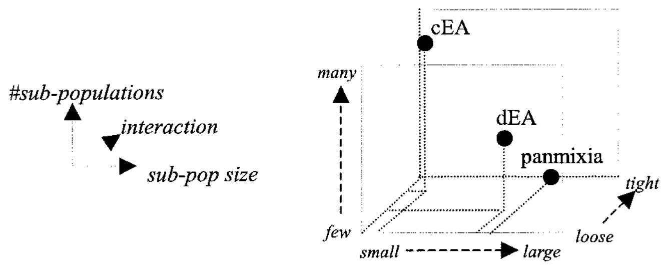


Fig. 4. Structured-population EA cube.

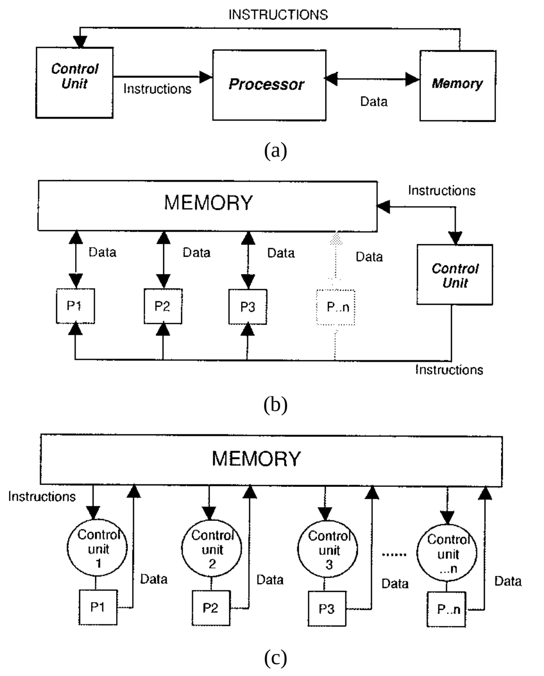


Fig. 5. Flynn's taxonomy for computer architectures. (a) SISD architecture. (b) SIMD architecture. (c) MIMD architecture.

While a dEA has a large subpopulation, usually much larger than one individual, a cEA has typically one single individual in every subpopulation. In a dEA, the subpopulations are loosely coupled, while for a cEA they are tightly coupled. Additionally, in a dEA, there exist only a few subpopulations, while in a cEA there is a large number of them. We use this cube to provide a generalized way for classifying structured EAs; we include panmictic EAs in it to clarify the figure. However, the points in the cube indicating dEA and cEA are only "centroids"; this means that we could find or design an algorithm that can be hardly classified as belonging to one of such two classes of structured EAs because its depends strongly on the selection of the values in each axis of the algorithm.

A. Nonstandard Structured EA Models

So far, we have made the implicit hypothesis that the genetic material, as well as the evolutionary conditions, such as selection and recombination methods, were the same for all the individuals and all the populations of a structured EA. Let us call these algorithm types *uniform*. If one gives up some of these constraints and allows different subpopulations to evolve with different parameters and/or with different individual representations for the same problem, then new distributed algorithms may arise. We will name these algorithms *nonuniform* parallel or dEAs. Tanese did some original work in the field and was the first in studying the use of different mutation and crossover rates in different populations [125]. Another more recent example of this class is the *injection island GA* (iiGA) of Lin *et al.* [78]. In an iiGA, there are multiple populations that encode the same problem using a different representation size, and thus different resolutions in different islands. The migration rules are also special in the sense that migration is only one-way, going from a low- to a high-resolution node. According to Lin *et al.*, such a hierarchy has a number of advantages with respect to a standard island algorithm. A similar hierarchical topology approach has been recently used in [112] with some differences such as real-coded GAs and two-way migration. The purported advantages are: no need for representation conversion, better precision, and better exploration of the search space using a nonuniform mutation scheme.

A related proposal has been offered by Herrera *et al.* [66]. Their *gradual distributed real-coded GA* involves a hierarchical structure in which a higher level nonuniform distributed GA joins a number of uniform distributed GAs that are connected among themselves. The uniform distributed GAs differ in their exploration and exploitation properties due to different crossover methods and selection pressures. The proposed topology is the cube-connected cycle in which the lower level distributed GAs are rings at the corners of the cube and rings are connected at the higher level along the edges of the cube. There are two types of migration: a local one among subpopulations in the same lower level distributed GA and global migrations between subpopulations belonging to different lower level distributed GAs. According to [65], the proposed scheme outperforms other distributed GAs on the set of test functions that were used in the paper.

In biological terms, different semi-isolated populations in which evolution takes place at different stages and with different, but compatible, genetic material makes sense: it might also be advantageous for EAs. Genetic programming, in particular, might benefit from these ideas, since the choice of the function and terminal sets is often a fuzzy and rather subjective issue.

Another class of nonstandard PEAs is given by *coevolutionary* approaches. The most well-known example is Hillis' method for finding minimal sorting networks by simulated evolution [67]. Hillis' approach consisted in a massively parallel cellular algorithm in which there are two independent pools or populations, each evolving according to a standard GA on the same grid. One population, the "hosts," represents sorting networks, while the other pool, the "parasites," represents test cases. The fitness of the sorting networks is given by measuring how well they sort the test cases provided by the parasites at the same grid location. Conversely, the parasites are scored according to how well they find flaws in the corresponding sorting networks. An "arms race" thus takes place that is beneficial for the evolution of the pseudoecology, resulting in exceptionally good sorting networks, very close to the best-known hand-made solution.

Another interesting coevolutionary variation of the cellular model is Sipper's *cellular programming algorithm* [114]. Cellular programming has been used extensively to evolve cellular automata for performing computational tasks and is based on the topological coevolution of neighboring cellular automata rules.

V. PEAs

In this section, our goal is to present a structured vision of the architecture, software tools, and parallel implementations of EAs. Therefore, Sections V-A to V-C are devoted to these three issues. Section V-E, discussing speedup and PEAs, is included because of the relevance of this measure in any domain relating to parallel algorithms.

A. Parallel Computer Architectures: An Overview

Parallelism can arise at a number of levels in computer systems: the task level, instruction level, or at some lower machine level. The standard model of Flynn is still widely accepted for classifying computer architectures [69]. But the reader should be aware that it is a coarse-grain classification. For instance, even today's serial processors are, in fact, parallel in the way in which they execute instructions, as well as with respect to the memory interface and many parallel architectures are hybrids of the base classes. For a comprehensive treatement of the subject, the reader is referred to Hwang's text [69]. Flynn's taxonomy is based on the notion of instruction and data streams. There are four possible combinations, conventionally called single instruction, single data stream (SISD), single instruction, multiple data stream (SIMD), multiple instruction, single data stream (MISD), and multiple instruction, multiple data stream (MIMD). Fig. 5 schematically depicts the three most important model architectures.

The SISD architecture corresponds to the classical monoprocessor PC or workstation.

In the SIMD architecture, the same instruction is broadcast to all processors. The processors operate in lockstep, executing the given instruction on different data stored in their local memories (hence the name single instruction, multiple data). The processor can also remain idle, if this is appropriate. SIMD machines exploit spatial parallelism that may be present in a given problem and are suitable for large, regular data structures. If the problem domain is spatially or temporally irregular, many processors must remain idle at a given time step, with the ensuing loss in the amount of parallelism that can be exploited.

In the MIMD class of parallel architectures, multiple processors work together through some form of interconnection. Different programs and data can be loaded into different processors, which means that each processor can execute different instructions at any given point in time. Of course, the processors will usually require some form of synchronization and communication in order to cooperate on a given application. This class is usually the most useful one, and most commercial parallel and distributed architectures belong to it.

There has been little interest in the MISD class since it does not lend itself to readily exploitable programming constructs. Systolic arrays belong to this class and are used in specialized applications such as signal processing.

Another important design decision is whether the system memory spans a single address space or it is distributed into separated chunks that are addressed independently. The first type is called *shared memory*, while the latter is known as *distributed memory*. This is only a logical subdivision that is independent of how the memory is physically built. In shared memory multiprocessors, all data are accessible by all the processors. This poses some design problems for data integrity and efficiency. Fast cache memories next to the processors are used in order to speed up memory access to often-used items. Cache coherency protocols are then needed to ensure that all processors see the same value for the same piece of data.

Distributed memory is also a popular architecture for multicomputers, which is well suited to most parallel workloads. Since the address spaces of each processor are separate, communication among processors must be implemented through some form of message passing. Networked computer clusters belong to this class. The drawbacks are that parallel computing performance is limited by high communication latencies and by the fact that the machines may have different workloads and are possibly heterogeneous. But these problems can be overcome, to a large extent, by using networked computers in dedicated mode with a high-performance communication network. In addition to the well-known fast-Ethernet networks, many other communication networks belonging to this class of low-latency high-performance systems can be created by using ATM or even Myrinet networks, allowing fast bit-transfer rates (150 Mb/s and 2 Gb/s, respectively).

Finally, we should not forget that the World Wide Web provides important infrastructures for distributed computation. Harnessing the Web or some other geographically distributed computer resource so that it looks like a single computer to the user is called *metacomputing*. The concept is attractive, but many challenges remain (see Section V-B).

Instead of describing the many special architectures and programming constructs that have been used for PEAs, in Section V-B we center the discussion on those parallel and distributed programming models that are independent of the particular hardware architecture, essentially those based on message passing in distributed memory architectures.

B. Tools for Parallelizing an EA

Here, we review some popular communication tools for implementing a PEA. Almost all the PEA toolkits are implemented by using the message passing model of communication, since it naturally allows profiting from multicomputers having distributed memory which are the most usual hardware found in universities and research institutes.

In the message passing model, processes in the same or on physically different processors communicate by sending messages to each other through a communication medium, such as a standard or specialized network connection. The two basic primitives are *send* and *receive* routines. In its simplest form, *send* specifies a local data buffer that is to be transmitted, and a receiving process identifier, usually in a different address space than that of the sending process. A *receive* operation usually specifies the sending process and a local data buffer into which the incoming data is to be stored. Together, a *send/receive* pair effects a memory-to-memory copy and an implicit process synchronization operation.

We include in the discussion basic tools such as sockets and more sophisticated environments, such as parallel virtual machine (PVM), message passing interface (MPI), Java, common object request broker architecture (CORBA), and Globus that provide a much richer set of functionalities than a simple message passing service. Section V-B9 is devoted to compare them from the point of view of PEAs.

1) Sockets: The BSD socket interface (see e.g., [31]) is a widely available message-passing programming interface. A set of data structures and C functions allow the programmer to establish full-duplex connections between two computers with TCP for implementing general purpose distributed applications. Also, an unreliable service over UDP (and directly over IP) is available for applications needing such a facility. Synchronous and asynchronous parallel programs can be developed with the socket application programmer's interface (API), with the added benefits of common availability, high standardization, and complete control over the communication primitives. But programming with sockets is error-prone and requires understanding low level characteristics of the network. Also, it does not include any process management, fault tolerance, task migration, security options, or other features usually requested by modern parallel applications.

2) PVM: The PVM [119] is a software system that supports the message-passing paradigm and permits the utilization of a heterogeneous network of parallel and serial computers as a single general and flexible concurrent computational resource.

PVM offers a suite of user interface primitives for communication and other chores. Application programs are composed of *components* that are subtasks at a moderately large level of granularity. The advantages of PVM are its wide acceptability and its heterogeneous computing facilities, including fault-tolerance issues and interoperability [120]. Managing a dynamic collection of potentially heterogeneous computational resources as a single parallel computer is the real appealing treat of PVM. Despite its advantages, PVM has recently begun to be unsupported (no further releases), and users are shifting from PVM to more efficient paradigms, such as MPI.

3) MPI: The MPI is a library of message-passing routines [45] similar to PVM but more complete. The MPI API was defined in the mid-1990s by a large group of people from academia, government, and industry. The interface reflects people's experiences with earlier message-passing libraries, such as PVM. The goal of the group was to develop a single library that could be implemented efficiently on the variety of multiple processor machines. MPI has now become a *de facto* standard, and several implementations exist, such as MPICH and LAM/MPI.[2]

The MPI functions support process-to-process communication, group communication, setting up and managing communication groups, and interacting with the environment. In addition, wide-area-network (WAN) facilities will be provided for MPI users in the near future.

4) Java-Remote Method Invocation (RM)I: The implementation of remote procedure calls (RPC) in Java is called Java-RMI [70]. RPC allows invoking of remote services in a transparent fashion and, therefore, is a natural step for programmers having "sequential" habits. The RMI in Java allows an application to use a remote service with the added advantages of being platform-independent and able to access the rest of the useful Java characteristics when dealing with distributed computing and the Internet in general.

The client/server model used by Java-RMI is, however, somewhat slow in the current implementations of Java, at least for scientific calculations, an especially important consideration when dealing with optimization algorithms.

5) CORBA: Although many systems currently deal with multithreaded, parallel, and/or distributed programs, some higher level research is devoted to how to glue together existing or future applications so they can work seamlessly in a distributed environment. Software systems that provide this glue have come to create the term "middleware," from which CORBA is one of the best known examples [129]. CORBA is based on object-oriented technologies. CORBA is a collection of specifications and tools to solve problems of interoperability in distributed systems.[3]

CORBA is especially important because it is growing on the application side rapidly; it allows clients to invoke operations on distributed objects without concern for object location, programming language, operating system, communication protocol, or hardware.

6) Globus: Globus is a new, extremely ambitious project to provide a comprehensive set of tools for building metacomputing applications [46]. Globus thus builds upon, and vastly

[2][Online] Available: http://www.mcs.anl.gov/mpi/mpich and http://www.mpi.nd.edu/lam

[3][Online] Available: http://www.omg.org

extends, the services provided by earlier systems such as PVM, MPI, and Legion [60].

The Globus toolkit consists of several modules that are used to help to implement high-level application services. One such service is what is called an Aadaptive wide-area resource environment (AWARE). It will contain an integrated set of services, including "metacomputing enabled" interfaces to an implementation of the MPI library, various programming languages, and tools for constructing virtual environments (CAVEs).[4]

7) OpenMP: OpenMP is a set of compiler directives and library routines that are used to express shared-memory parallelism.[5] The OpenMP API was developed by a group representing the major vendors of high-performance computing hardware and software. Fortran and C++ interfaces have been designed, with some efforts to standardize them. The majority of the OpenMP interface is a set of compiler directives. The programmer adds these to a sequential program to tell the compiler what parts of the program to execute concurrently, and to specify synchronization points. The directives can be added incrementally, so OpenMP provides a path for parallelizing existing software. This contrasts with the Pthreads and MPI approaches, which are library routines that are linked with and called from a sequential program, and which require the programmer to manually divide the computational work.

8) Internet-Based PEAs: To support the idea of the rapid evolution of PEA's technologies, we are including this section to deal with the new issues relating PEAs and the Internet as the new natural platform for computing. Basically, the Internet is a new domain for PEAs characterized by: 1) the large amount of available (dedicated or not) machines; 2) large heterogeneity of operating systems and configurations; and 3) the set of communication protocols, access restrictions, and failure conditions that need to be taken into account.

Some of the considerations for running a PEA in a general WAN also hold in the Internet domain, but there are other added difficulties for performing PEAs in Internet. Some of them are the access and potential failure of geographically distant machines computing in parallel and the dynamic load balancing needed by some solution techniques. Also, there exists considerable influence of the Internet features on the numeric behavior of the algorithms and the difficulty of implementing really efficient algorithms in Internet. The latter is a challenging research line: researchers must report not just that an algorithm is viable, but also address its competitiveness with other algorithms.

Despite the numerous problems, some Internet algorithms exist that are facing such tasks for the first time. For example, in [27], the author develops and tests a distributed GP (DGP) program written in Java with an architecture specially devised to deal with Internet. DGP has been tested by running, in parallel, clients and servlets on heterogeneous machines available on tje Internet; also, it has been compared with traditional GP and LAN-distributed GP on various problems. A free distribution exists with graphical interface and the preliminary results show that this alternative is feasible and even performs a better search than other techniques on the tested problems.

[4]See [47] and the Globus Web site for detailed information and current status of the project. [Online] Available: http://www.globus.org

[5][Online] Available: http://www.openmp.org

Also, much still must be said as to new Internet technologies. We here spread some seminal ideas about what the new trends will be. First, involving Extensible Markup Language (XML) in algorithm definition (parameters) and information exchanges is a clearly desirable use of new technologies. This will help in unifying different implementations and could allow different researchers to interact through generic XML connectors.

Next, there is an obvious extension of present PEAs with respect to new languages such as C# and J# (the .NET version of Java). Deploying existing algorithms in these platforms is a clear way of achieving larger impact in final users with Microsoft Windows technology. Finally, using some of the multi-language support for*Simple Object Access Protocol (SOAP)* will help in developing optimization repositories in Internet for remote access to developed algorithms.[6] Some of these ideas are being used in our own and other's projects,[7] while still much must be said about the efficiency and real use of these systems; in addition, some running projects such as DREAM[8] are developing high abstract environments for optimization and artificial life in the Internet.

9) Comments on These Communication Tools and PEAs: Although a number of PVM and socket implementations exist for PEAs, future trends will have to deal with the Internet as a pool of computational resources. Many problems arise when a PEA has to be implemented on a heterogeneous WAN linking computers with different operating systems. Some of these requirements are:

- transparent exchanges of data types in the Internet;
- remote execution and tracing;
- security issues (firewalls, proxies, etc.) relating access, data communication, and process execution;
- process migration between clusters for PEA applications which are time-consuming (real-world problems);
- adaptation to the dynamic behavior of the links and computers involved in the execution of the PEA;
- fault tolerance.

All these requirements naturally lead to using Java, MPI, and, maybe in the future, Globus. Despite the advantages of Java for multiplatform execution, communication, and related tasks, the present execution speed of Java for scientific computations could be a drawback for some PEA applications. In fact, there exists some PEA systems such as MARS [124] and the the MALLBA library[9] that account for some of these requirements. A great deal of information on networking and EAs can be found in the NEO server.[10]

We think that MPI is currently the better choice for implementing large-scale and general-purpose PEAs because of its present and future advantages. In the case that Globus will dominate the metacomputing field, MPI has a bridge to access Globus services. Other communication models such as the one offered by CORBA are also important for implementing PEAs, since computations can be coded in C++ and sophisticated services

[6]For more on Internet technologies: [Online] Available: http://www.w3.org

[7][Online] Available: http://neo.lcc.uma.es and http://geneura.ugr.es

[8][Online] Available: http://www.dcs.napier.ac.uk/benp/dream/dream.htm

[9][Online] Available: http://www.lsi.upc.es/mallba

[10][Online] Available: http://neo.lcc.uma.es

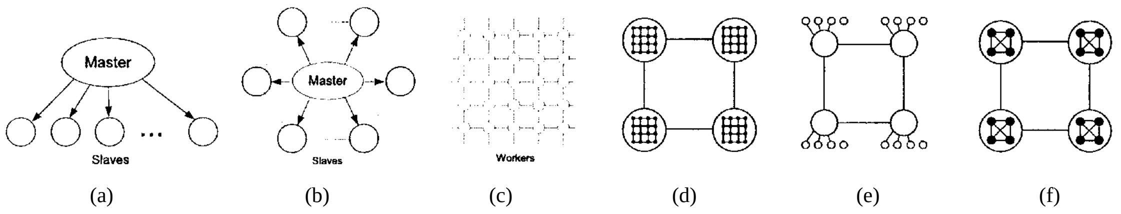


Fig. 6. Different models of PEA: (a) global parallelization; (b) coarse grain; and (c) fine grain. Many hybrids have been defined by combining PEAs at two levels: (d) coarse and fine grain; (e) coarse grain and global parallelization; and (f) coarse grain at the two levels.

can be directly and transparently managed by the object broker system. Although Java-RMI and CORBA provide a very good level of flexibility for PEAs, they are still far from allowing scientific efficient programs to run in a competitive time with C or even C++.

At present, new domains and potentially new research lines are being open because of new protocols such as SOAP for remote method invocation and message passing, or the Microsoft .NET environment, with new interesting programming languages such as C#. Also, integrating XML in PEAs for exchanging data or configuring the algorithms are new possibilities which have become available only in the past two or three years.

Nonetheless, regardless of the degree of sophistication of the tool, each user potentially has different necessities, and thus many users can still benefit from using PVM or other communication libraries for locally managed clusters. In fact, we can find PEAs implemented with almost any of the mentioned parallel programming paradigms, in particular using the more traditional MPI, sockets, and PVM.

C. Parallel Implementations of EAs

Whichever computer and software are being used to build a PEA, a researcher usually pursuits several goals. The expected advantages coming from the parallelization of EAs can be summarized as follows:

- finding alternative solutions to the same problem;
- parallel search from multiple points in the space;
- easy parallelization as islands or neighborhoods;
- more efficient search, even when no parallel hardware is used;
- higher efficiency than sequential EAs, in many cases;
- easy cooperation with other (EA and traditional) search procedures;
- speedup due to the use of multiple CPUs (faster search).

These advantages contrast with the features of other heuristic algorithms which usually impose artificial constraints, which search from only one point to another new point, or which find one single solution at a time, and have a higher probability of getting stuck in local optima than EAs.

Now, let us go back to the study of existing implementations of PEAs. Some of the most well-known parallel implementations are depicted in Fig. 6. *Global parallelization*, shown in Fig. 6(a), provides a panmictic-like evolution with evaluations performed in parallel. This is faster than a sequential panmictic EA and does not require load balancing except in the case of GP, where individuals to be evaluated can be of very different complexity, or in special cases such as some robotics applications, where the evaluation cost of a solution is not constant. Genetic programming can be efficiently parallelized at the fitness-evaluation level by using steady-state reproduction instead of the usual generational algorithm [11], [28]. (See [130] for an example of global parallelism in ES.) Another example of the use of global parallelism for evolving artificial neural networks with EP is given in [103]. The efficiency of global parallelization has been analyzed in [96] for GP and, in a very thorough manner, by Cantú-Paz for GAs [22], where the model is called a master–slave.

Automatic parallelism is not presented in this paper, since compilers introducing parallel search in a sequentially written algorithm are rare in the EC field. Nonetheless, some automatic parallel Fortran such as Vienna Fortran [18] could be used to provide parallel execution of sequential EAs.

Also, many researchers use a pool of processors to speed up the execution of a sequential algorithm, just because *independent runs* can be made more rapidly by using several processors than by using a single one. In this case, no interaction at all exists between the independent runs. This extremely simple method of doing simultaneous work can be very useful. For example, it can be used to run several versions of the same problem with different initial conditions, thus allowing gathering statistics on the problem. Since EAs are stochastic in nature, being able to collect this kind of statistic is very important.

Neither of the above adds anything new to the nature of the EAs, but the time savings can be significant. In Fig. 6(b), we can also see the popular multipopulation or *island model*, in which several EA subalgorithms run in parallel in a connected mode (by exchanging individuals during their search). This model can have a MIMD implementation in a straightforward way, by mapping every island to one processor. New parameters such as the number of individuals being exchanged, the frequency of migration, the selection/replacement of individuals, and the topology need to be set. Once again, Tanese performed some of the first measurements of the influence of migration frequency and number of individuals exchanged [126]. (See [7], and [22] for further indications of the values to be used for several heterogeneous problem benchmarks in GAs and [43] for GP.)

The *cEA*, shown in Fig. 6(c), is usually parallelized on SIMD machines, since the individual-processor mapping is quite direct [128], [94]. However, nothing prevents one from using a single processor for executing a cEA [6], [54], nor an MIMD implementation [84], [115].

Fig. 6(d)–(f) shows three *hybrid algorithms* in which a two-level approach of parallelization is undertaken. In the three cases the higher level for parallelization is a dEA (coarse-grain implementation). The model shown in Fig. 6(d) was initially used by

[63] and has received some attention in works like [7], [8], [56], and [79]. In this model, the basic islands perform a cEA, thus trying to get the combined advantages of the two models. In Fig. 6(e) (see, e.g., [49]), we have many global parallelization farms connected in a distributed (migration) fashion, thus exploiting parallelism for making fast evaluations and for obtaining separate population evolution at the same time. Finally, in Fig. 6(f), we present several farms of distributed algorithms with a still higher level dEA making migrations among connected farms. Although these combinations may give rise to interesting and efficient new algorithms, they have the drawback of having even more new parameters to account for the more complex topological structure. (See details on this and the rest of models also in [21].)

As we have seen, both panmictic and structured models can be parallelized in many ways, and can be implemented on monoprocessor machines. The classical relationship between distributed/cEAs and coarse-/fine-grain parallelization is thus an artificial one to a large extent, and need not concern any longer.

D. Measuring the Speedup Is Controversial

Computing the speedup of a parallel algorithm is a well-accepted way of measuring its efficiency. Although speedup is very common in the deterministic parallel algorithms field, it has been adopted in the PEA's field in different flavors, not all of them with a clear meaning. In this section, we will present speedup, discuss its meaning, and several applications for measuring the efficiency of a given PEA.

Our goal is to review and discuss the existing works whose contents are relevant for studying the speedup in PEAs (see Table. I). Let us begin first by revisiting its traditional definition. The well-known definition of speedup (see e.g., [3]) relates the (worst) execution time of the best sequential version T_1 to the (worst) execution time of the parallel version of the algorithm being evaluated on m processors T_m

$$S_m = \frac{T_1}{T_m}.$$

With this definition, we can distinguish amongst

- sublinear speedup $S_m < m$;
- linear speedup $S_m = m$;
- super-linear speedup $S_m > m$.

The first modification which we need to introduce in the standard speedup definition is to consider average times in the ratio. The reason is that EAs are stochastic algorithms in which one single execution is not statistically significant. This means we need to average a given number of statistically independent runs in order to have representative time values

$$S_m = \frac{\bar{T}_1}{\bar{T}_m}$$

Even by using average times, the traditional definition remains unclear in the field of EAs, since it makes the assumption of being aware of the best algorithm to solve the problem. We will call it the *strong definition* of speedup (see Table I, type I). Some practical problems arise with this definition. First, it is difficult, if not impossible, to decide whether or not a sequential EA is the best algorithm, since many times it is the only existing algorithm being tried for the problem (e.g., new applications). Second, in analyzing EAs, it is usual to study a large set of problems; the strong definition requires the researcher to be aware of the faster algorithm solving any of the problems being tackled. This scenario is often not a realistic situation.

TABLE I
TAXONOMY OF SPEEDUP MEASURES

I. STRONG SPEEDUP
II. WEAK SPEEDUP
A. SPEEDUP WITH SOLUTION-STOP
1. VERSUS PANMIXIA
2. ORTHODOX
B. SPEEDUP WITH PREDEFINED EFFORT

These reasons had traditionally led researchers to measure the speedup by comparing their own sequential and parallel algorithms. We will define a *weak definition* of speedup (see Table I, type II) as the extent to which it is possible that a different algorithm exists (probably not an EA) that solves the problem faster in sequential mode. This definition will allow us to compare our PEA against well-known sequential EAs, therefore studying the speedup without needing to involve nonEAs in the analysis.

The next important point relating a weak definition is the stopping criterion. Speedup could be studied by imposing a *predefined global number of iterations* both to the sequential and to the PEA. We call this a measure of type II.B (Table I). In general, we dislike this kind of measure, since it compares two algorithms that are working out solutions of different fitness (quality), thus breaking the fundamental statement of being "solving" the same problem with the "same" precision. This stopping criterion can be useful in some other situations where, e.g., the same effort is allocated to different algorithms to compare their final error, but not when speedup is to be measured. Important papers in this field such as [64] and [23] also make the same considerations we have just pointed out.

Therefore, we need a meaningful and fair termination criterion. The obvious candidate is to stop the comparison of algorithms when a solution of the same quality had been found, usually an optimal solution. We call this an *orthodox weak* definition or type II.A.2 (Table I).

Let us gain a deeper understanding of the orthodox weak definition. One important consideration is the composition of the sequential EA. By following the old-fashioned concept that a "sequential" EA is a "panmictic" EA, we would compare a panmictic (sequential single population) EA with, e.g., a dEA of d islands, each one running on a different processor. We call this a *versus panmixia weak* comparison (Table I, type II.A.1). But, the algorithm running in one processor is panmictic in this case, while the d islands that are using d processors represent a distributed migration model whose algorithmic behavior is quite different from the panmictic one. This could sometimes provoke a very different result for the numerical effort needed to locate the solution, and thus very different search times can be obtained (in general, faster search for the distributed version). In fact, it could lead to obtain a super-linear speedup of a considerably high magnitude, since the dEA running parallel islands can locate a solution more than d times faster than the panmictic one [8], although this need not always be the case, as has been shown by Punch for certain problems in parallel genetic programming [99].

In order to have a fair and meaningful speedup definition for PEAs, we need to consider exactly the same algorithm (for example, the distributed one with d islands) and then only change the number of processors, from 1 to d, in order to measure the speedup (*orthodox weak definition*). In any case, the speedup measure should be as fair and as close to the traditional definition of speedup as possible.

In addition, we must mention an obvious result, namely that adding more processors is not always more efficient for any parallel system. Only some models are scalable (more efficient) with an increasing number of processors.

We now present a striking point. Many authors have analyzed PEAs attending to different criteria, and many of them came out with super-linear speedup when using a parallel machine [12], [17], [113], [62]. After having discussed alternative methods to measure the speedup we have still to address one question: is it really possible, to get super-linear speedup in PEAs? The answer to this question is *yes*. In short, the sources for super-linear speedup are (see [4] for more details):

- the higher chances of finding an optimum by using more processors, due to the intrinsically heuristic multipoint nature of PEAs;
- splitting the global large population into smaller subpopulations that fit into the processor caches provides faster algorithms than using a single main memory;
- the operators work on much smaller data structures, and they do so in parallel, which is an additional source of speedup.

VI. Empirical Behavior of Parallel EAs

In this section, we summarize some important results for PEAs from a practitioners' point of view. Our aim is to offer some guidelines for researchers dealing with structured EAs, either parallelized or not.

Let us begin with dEAs. Many approaches have been employed in order to discover the best topology for a multipopulation algorithm, especially when the island is a GA. The results are often inconclusive, since sometimes a hypercube is more efficient than a ring, and sometimes a mesh is more/less efficient than the previous ones [17], [24]. The same holds for the migration rate (although quite a small number of individuals seems the best approach) and the migration frequency (although infrequent migration seems to be the best choice). As noted before, leading work on migration rates and frequencies was performed by Tanese [126]. Some additional research has been conducted from a practical point of view for an assorted set of problems and distributed models (e.g., [7]), in which some parameters are set and some others are left free in order to change the behavior of the algorithm. For example, a ring is a good idea since migrations can be made in a constant and small time [23]. Also, by sending one single individual, we are free of tuning the migration frequency to achieve an efficient algorithm [7]. See also [43] and [42] for a consistent set of empirical results on distributed GP concerning the sizing of populations, topologies, and migration rates.

A heuristic conclusion (not valid in every case, of course) is that migrating the best individual in the island might be worthwhile for function optimization, although migration of a random individual can be preferable for combinatorial optimization and highly epistatic problems (parameter correlation), since premature convergence is not enforced in the distributed subalgorithms. This "conquest" effect is a hard problem in complex domains.

With respect to cEAs, it seems appropriate to use a rectangular grid instead of a square one as the default disposition of individuals for problems with intrinsic epistasis [6]. Square grids provide faster evolution for nonepistatic problems [10]. Also, introducing some kind of problem-specific local-search into the cEA is a good idea in order to quickly tune solutions in function optimization, and in general, in order to speed up the convergence, since the basic property of cEAs is that of maintaining a high level of diversity [91].

VII. Theory of PEAs

EC is a relatively novel research field in optimization, learning, and numerous application areas. Many advances have appeared concerning the theoretical foundations of EAs, especially for GAs and ES. However, the results are not conclusive yet and the area is still open.

In order to get a structured picture of the existing theoretical advances and future issues, we proceed to classify them into several subfields in which theory is especially important:

- representation theories;
- operator theories;
- structured algorithms;
- convergence theories;
- fitness landscape theories;
- unification theories;
- working model theories;
- speciation theories and niches.

Among these, structured algorithms, unification theories, and working model theories are directly related to PEAs. The rest can be linked to PEAs, as well as to any other EA.

We define *representation theories* to include all formal explanations that lead to an understanding of the behavior of a given genotype-phenotype mapping. The basic notion of *schema* [68] and *forma* [100] can be included here. These theories help in selecting more appropriate representations for the symbol strings of a given new application. These theories usually relate the behavior of the operators and fitness landscape to the kind of genotype in hands, as just stated in [37] and [44]. Studying the theoretical implications of using an assorted set of representations in a PEA is an open research line.

We call *operator theories* the formal characterizations of the work of variation operators such as recombination, mutation, local search operators, etc. Interesting results in this sense are the definition of uniform crossover [121], the role of mutation [15], and a framework for hybridizing EAs [33]. Operator theories to explain the behavior of the migration just as a new operator that maps the present set of search-points to next one are sought. Also, defining different sets of operators and/or control parameters for each of the component subalgorithms is an interesting open research line.

Works formalizing *structured algorithms* are especially important for the focus of this paper since there is an important link between using structured populations and running parallel algorithms. Defining how a panmictic algorithm can be extended to a multipopulation distributed algorithm, or how introducing neighborhood locality into the population can enhance the search, are two examples of such results. In multipopulation algorithms, some theoretical advances relating the optimum subpopulation sizes and coupling can be found in [24] and [50]. The similarities between the punctuated equilibrium evolution and the search in a multipopulation algorithm are stressed in [30], while other (experimental) results concerning the degree of interconnection depending on the kind of problem can be found in [7]. Also, the use of a cellular (neighborhood) model has been modeled in [108] to calculate the selection pressure, and in [25] to give some useful statistical measures for tracking and understanding the search of such an EA. A more detailed discussion of these issues is found in Sections VII-A and VII-B.

Some basic *convergence theories* exist for sequential EAs. They are generally useful for calculating convergence rates. Some especially interesting results can be found in relation to the response to selection $R(t+1)$ in [90], defined by the difference between the population mean fitness of two consecutive generations: $R(t+1) = M(t+1) - M(t)$. In their paper, the authors provided a very useful tool for analyzing EAs in terms of a new concept called *selection intensity*. Also, important milestones relating takeover regimes in traditional domains like GAs [51] can be included in this class. These theories must be extended to deal with convergence in PEAs, just like the results in [22] in relation to convergence times, deme size, or speedup. Also, an exceptionally interesting work unifying the study of panmictic and structured EAs through the use of hyper-graphs can be found in [117], which additionally helps in predicting takeover times and probabilities.

Some theories relating the *properties of the fitness landscapes* and the search of the EA have been proposed. For example, stochastic reverse hill-climbing allows incorporating into the algorithm explicit information about the landscape with the goal of using this information within some genetic operator [32]. Also, fitness landscapes have been classified, attending to several features in order to help researchers to known how difficult a problem is. In particular, the degree of epistasis in the individuals has been usually considered a criterion to rank problems [93]. The number of optima (multimodality) and the number of local optima in the landscape are two important factors determining the difficulty of a problem. Domains which require locating many optima at the same time [35], managing restrictions [29], dynamic fitness landscapes [52], or having time due dates [72] usually impose some requirements to the kind of operators and fitness measures being used. The information on the fitness landscape can be used statically by selecting the kind of EA beforehand, or alternatively it can be fed somehow into the algorithm at run time in order to adapt the PEA for searching in the more promising regions. In general, PEAs have been developed to cope with these problems, such as [110] for using cEAs in dynamic environments, some works dealing with multimodality and epistasis in dEAs [7] and cEAs [6], and the close relationship between the work of Daida *et al.* [33] on hybridization (i.e., adding problem knowledge to the EA) and granular EAs (i.e., structured EAs).

We refer to *unification theories* as the set of formal descriptions of separate EA models as if they were instances of a higher level metaheuristic. These theories allow, for example, unification of PEAs under a common formalization, showing that coarse-grain, fine-grain, structured, and panmictic EAs can be studied under a common point of view. An example of such theory is the mentioned unified description that Sprave has created for PEAs based on the concept of hypergraphs [117]. By defining the subpopulations (whatever their size is) as vertex in the hypergraphs and their relationships as the edges, structured and panmictic algorithms can be characterized in a natural way. Since PEAs are usually parallel implementations of structured EAs, the behavior of the algorithm can be studied. Furthermore, PEAs could be described after the basic notion of the Adaptive Granulated System, derived from the initial model of Holland [68] for an adaptive algorithm. In addition, unification models for the search with heuristic models in general, or for representation and operators in particular, help in knowledge exchanges [44].

We separately listed the *working models* from the rest of theories since, although they have much to do with them, the working models focus on providing executable versions of the behavioral run-time properties of EAs. Such a model for a sequential GA can be found in [132], where the bases for an executable model of a simple GA are discussed. More sophisticated and even exact models for sequential GAs can be found in [86]. In relation to PEAs, there are similar results that we will discuss in the following sections.

Finally, we must point out that the research performed on PEAs and on *speciation methods* share some similarities [35]. Speciation EAs aim to locate several optima at the same time. To achieve this goal, speciating algorithms explicitly maintain different solution species during the optimization process by applying specific techniques (e.g., *sharing*). PEAs lead naturally and implicitly to creating multiple niches of solution species, just as speciation EAs. However, the latter dynamically allocate a different number of trials and a different fitness value to individuals in every niche. This concentrates effort on more promising peaks, while still maintaining individuals in other areas of the search space. dEAs provide only constant-sized niches, and no fitness changes are associated to any standard parallel model. On the other hand, cEAs allow speciation, e.g., in a grid, but one particular species will eventually take over the whole population if no specific operators are included. Hence, PEAs can be used for multimodal optimization, but they usually need to be combined with specialized operations to deal with *niches* in their traditional acceptation in optimization.

A. dEAs

Because of established tradition in using distributed models of EAs, many works deal with theoretical issues for dEAs.

We can find these algorithms under different names, such as multipopulation EAs, coarse-grain EAs, island-model EAs, and others. Designing efficient dEAs and understanding their dynamical behavior are difficult problems due to the number of parameters and their nonlinear interactions which are not well understood. Therefore, it would be most useful to have some theory behind the setting of those parameters, a setting which is most often done by empirical trial and error.

Let us begin with early work relating dEAs and the building-block hypothesis. The work of Pettey and Leuze [97] was one of the first in stating that a PGA with uniform communications can be expected to allocate trials to schemata in a exponentially increasing (decreasing) fashion, i.e., in a manner consistent with Holland's original schema theorem for sequential GAs. They backed up these results by also undertaking practical application of their theory on De Jong functions [71] and the travelling salesman problem (TSP). The experiments were made with a hypercube multiprocessor containing eight nodes. In the same vein, Munetomo *et al.* [92] relate the decreasing values of the mean fitness of the island model to a decrease in the number of building blocks being processed in parallel. Therefore, they state the importance of maintaining diversity, and propose a special asynchronous migration model for enhancing building blocks management.

Although the main focus of the present review is on PEAs for search and optimization in science and engineering, and not on biological population modeling, it is worth mentioning another early work by Cohoon *et al.* [30], in which it is argued that the success that is often empirically observed in PGAs may be due to a phenomenon similar to punctuated equilibria. Punctuated equilibrium theory states that natural evolution is characterized by long periods of relative stasis, punctuated by periods of geologically rapid change associated with speciation events. According to Cohoon *et al.*, migration between demes can trigger such rapid evolutionary changes. A recent reference to work along these lines by the same group is [83].

The work from Goldberg [50] provided a firmer footing on optimal population sizes for both serial and parallel GAs. He used a figure of merit, the real-time rate of schema processing, in order to calculate the relationship between population size and the elapsed time until convergence for the population. His results suggested the use of small/large populations in serial/parallel implementations of GAs. Also, some recommendations were made for applying and extending the provided theory to problems not covered by the assumptions made in his study.

The most relevant work on the dynamics, convergence, and efficiency of parallel and distributed GAs in the last years has been done by Cantú-Paz and Goldberg. This groundbreaking work is summarized nicely in [22]. In relation to real-time measures for parallel GAs, the contributions in [24] predicting the scalability of parallel GAs are most appealing. First, they compute the optimal number of processors that can be used by different types of GAs in order to determine the optimal number of processors to minimize the execution time. Also, this work offers some bounds for the topology, rate, and frequency of migrations. They assume a model in which migration occurs after convergence of the subpopulations or at every generation, which is not the case in many applications and parallel models where migrations typically occur every few generations. However, they showed that the optimal number of processors n that minimizes the execution time is directly proportional to the square root of the population size μ and the fitness evaluation time T_f, and inversely proportional to the mean time to communicate with one processor T_c, as follows:

$$n = O\left(\sqrt{\frac{\mu T_f}{T_c}}\right).$$

This formally states that many practical applications can benefit from using large numbers of processors, since these two factors increase as the domain becomes more difficult. The analysis also shows that simply distributing the available individuals into multiple islands without communication does not offer a significant improvement over a panmictic population.

As for the choice of the communication topology, the conclusions of the model are that islands with many neighbors are more effective than sparsely commected demes. This brings forth a tradeoff between computation and communication. Following the model, optimal choices of the degree of the topology that minimizes the total cost can be made [22].

Cantú-Paz also treats the often neglected effect of the choice of migrants and the individual replacement policy and shows that choosing the migrants and replacing according to fitness increases selection pressure and accelerates convergence. All these results, though pertaining to PGAs, should prove useful to pave the way for a more principled study of other PEAs as well.

A work from Niwa and Tanaka [95] performs a Markov chain analysis based on the Wright–Fisher model found in population genetics. The model leads to the computation of the mean convergence time under genetic drift, which is found to be proportional to the population size, with the coefficient being larger with smaller migration rates. Besides and especially, they derived the most effective migration rate for a simplification of the island model GA. In fact, they propose to send one single individual for each generation. They also show that the distributed GA is not only better in managing larger population sizes than the panmictic one, but also in keeping the diversity in population better than usual GAs.

Speedup and convergence results are obviously important, but another fundamental area of research is the characterization of the class of problems for which the use of multiple populations would be beneficial. Whitley *et al.* [134] have presented an abstract model along these lines and made experiments of when one might expect the island model to outperform single population GAs on separable problems. Linear separable problems are those problems in which the evaluation function can be expressed as a sum of independent nonlinear contributions. The authors find indications that partitioning the population may be advantageous in this case. This is a good starting point, but much remains to be done in this area.

B. cEAs

cEAs are a kind of stochastic cellular automata [131], [128] where the cardinality of the symbol alphabet is equal to the number of points in the search space. To our knowledge, the only theoretical model of cEAs is the one proposed by Rudolph and Sprave. In [106], they show how cellular GAs can be modeled by a probabilistic automata network and give a proof of complete convergence to a global optimum based on Markov chain analysis. This result was obtained under the assumption that each individual's fitness value in the grid has to be better than a certain threshold which depends on past fitness values and on the current generation number. Local fitness-proportionate selection was used for reproduction, which is not the customary choice for cellular GAs, but has some theoretical advantages. It was also remarked that rather large neighborhood sizes (tens of grid points) were needed for good behavior in the 1–D (ring) case, while subsequent work on 2-D grids (torus) [107] showed that the ideal neighborhood size is much smaller, of the order of five, which confirms the empirical findings of other studies.

Statistical and empirical analyzes of the dynamical behavior of cGAs have been performed by Sarma and De Jong [108] and by Capcarrère *et al.* [25]. Sarma and De Jong's work concentrated on the effect of the neighborhood's size and shape on the selection pressure and on the local selection method. In [108], De Jong and Sarma studied the influence of choosing a particular selection method for use in cellular GAs. Due to the lack of a good model of the dynamics of cellular GAs operating with overlapping neighborhoods, this study is necessarily empirical, making use of repeated measures of the relevant quantities on a number of well-known test functions. Three standard selection algorithms were used: fitness-proportionate, linear ranking, and binary tournament. The cellular GA structure was a 2-D toroidal grid of size 32×32 with three different neighborhood shapes with five, nine, and 13 neighbors respectively, which are the most common in practice.

In order to study only the induced selection pressure (without introducing the perturbing effect of recombination and mutation operators) a standard technique is to let reproduction be the only active operator and to monitor the growth rate of the best individual in the initial population. A first remark is that when we move from a panmictic population to a spatially structured one of the same size, the global selection pressure induced on the entire population is qualitatively similar but weaker. In the spatially distributed case it was observed that for all three mentioned neighborhoods the performance of fitness-proportionate selection was inferior to that of linear ranking and binary tournament, with binary tournament being roughly equivalent to ranking as the neighborhood size increases.

To understand why this is so, De Jong and Sarma, using the same initial population for the three selection methods, monitored the actual number of offspring produced by each member of the population. The experiences were repeated a large number of times with different random number seeds in order to estimate the variance due to sampling errors. It was found that fitness-proportionate selection induces a much uniform but weaker global selection pressure than either ranking or binary tournament selection. It is well-known that ranking and binary tournament produce a constant selection pressure independent of the actual fitness values, while proportionate selection is obviously quite sensitive to actual fitness values. In the case of cEAs with small local selection pools, it appears thus that local selection methods based on ranking or binary tournament offer superior performance. In terms of the tradeoff between efficiency and simplicity, De Jong and Sarma conclude that local binary tournament combined with an elitist policy for the replacement of an individual seems to offer the best solution, and this is in qualitative agreement with what other researchers have been doing empirically in the past.

In a subsequent study, Sarma and De Jong [109] performed a more detailed empirical analysis of the effects of the neighborhood's size and shape on the local selection algorithms. Five different neighborhoods with up to 13 neighbors were taken into account and two selection methods were used: fitness-proportionate and linear ranking. Again, the growth rate of the best individual was studied using a 2-D toroidal grid of size 32×32. As in the previous case, it was found that the global selection pressure has the same qualitative behavior in the panmictic case as well as in the grid, but it is weaker in the structured population, with linear ranking being stronger than fitness-proportionate in both cases. The difference in intensity was attributed by Sarma and De Jong to the finite propagation speed of the individuals through the grid in the spatially structured case. In fact, they were able to show that propagation times are closely related to the neighborhood size, with larger neighborhoods giving rise to stronger selection pressures. However, they also found that two neighborhoods having the same linear extension but a different number of neighbors show nearly identical results.

In conclusion, these studies have put the choice of some critical parameters of the cellular model on a firmer and more systematic basis, whereas previous work had been empirical and tentative in character. Along the same lines, the work of Alba and Troya [6] discusses the effect of the ratio depending on the kind of problem, as well as they propose a tunable cEA in which the ratio between the radii can change during evolution. It is worth noting that a similar study has been recently done by Gorges–Schleuter [58] for ES.

Capcarrère *et al.* defined a number of statistical measures that are useful for understanding the dynamical behavior of cEAs. Two kinds of statistics were used: *genotypic* and *phenotypic*. Genotypic measures embody aspects related to the genotypes of individuals in a population. Phenotypic statistics concern properties of individual performance, essentially fitness (see [25] for the exact definitions).

Among the genotypic statistics, quantities that measure how individuals are spatially distributed were defined. One of these is the frequency of transitions, which is equal to the number of borders between homogeneous blocks of cells having the same genotype divided by the number of distinct couples of adjacent cells. In practice, this measures the probability that two adjacent individuals have different genotypes. To measure the genotypic diversit,y the customary population entropy was used, as well as a diversity index, defined as the probability that two randomly chosen individuals have different genotypes. It was shown that there is a relationship between the diversity index and the previously defined frequency of transitions indicator: the diversity index is just the expectation value of the frequency of transitions.

Phenotypic statistics deal with properties of phenotypes, principally fitness. The performance of a population is defined as its average fitness. The diversity at the phenotypic level is customarily defined as the variance of its fitness distribution and its definition is identical in cellular algorithms and in panmictic EAs. More interesting are the new measures that apply in the cellular case and that do not have a counterpart in the global case. Capcarrère *et al.* defined a new metric called the *ruggedness*, which measures the dependency of an individual's fitness on its neighbor's fitness, giving the fitness correlation between neighboring sites.

VIII. New Trends in PEAs

In this section, we focus on some of the most promising research lines in the field of PEAs. Future achievements should take note these issues.

- *Tackling dynamic function optimization problems (DOP)*: PEAs will have an important role in optimizing a complex function whose optima vary in time (learning-like process). Such problems consist in optimizing a successive set of fitness functions, each one usually being a (high/small) perturbation of the precedent one. Industrial processes, such as real task-scheduling, and daily life tasks such as controlling an elevator or the traffic light system, can be modeled by dynamic systems. Some PEAs, like cEAs [110] and dEAs, can deal with such DOP environments successfully thanks to their natural diversity enhancements and speciation-like features. Since structured EAs keep diversity high, they can react quickly to a change in the environment (optimum location). At the moment of the environmental change, a new fitness function is now to be optimized, and PEAs will find diverse genotypic material to redirect the search toward the new optimum location. This is a natural advantage of these algorithms with respect to most sequential and/or panmictic EAs.
- *Developing theoretical issues*: Improving the formal explanations on the influence of parameters on the convergence and search of PEAs will endow the research community with tools allowing to analyze, understand, and customize an EA family for a given problem. Advances relating the number and size of subpopulations and the degree of connectivity between the subalgorithms or neighborhoods will have a special interest. In addition, extending traditional deterministic measures and results to analyze stochastic PEAs will endow this EA branch with a more serious set of tools to characterize them.
- *Relationship between PEAs and other nonEAs*: This is a large area that is important in practice. Comparing PEAs with other search heuristics will allow to interface algorithms coming from different users and with different search properties to work together in order to solve a single complex task in parallel. Of course, there is no *a priori* reason for an EA to be superior to another search technique; and this clearly advocates to employ hybrid techniques to obtain a better algorithm. This is very important from a theoretical and practical point of view, since hybridizing and parallelizing are two growing trends in solving complex problems.
- *Running PEAs on geographically separated clusters*: This will allow a user to utilize sparsely located computational resources in a metacomputing fashion in order to solve his/her optimization problem. A distinguished example of such a system is to use the Web as a pool of processors to run PEAs for solving the same problem.
- *Benchmarking soft computing techniques*: At present, it is clear that a widely available and large set of problems is needed to assess the quality of existing and new EAs. Problem instances of different difficulty specially targeted to test the behavior of EAs and related techniques can greatly help practitioners in choosing the most suitable EA or hybrid algorithm for the task at hand. Some important classes of problems show epistasis, multimodality, nonstationarity, constrains, and learning processes, all of which are important components of real-life problems.

IX. Classification of PEA Implementations

In this section, we discuss briefly the main features of some PEAs by presenting a structured classification from three points of view: by model, by type, and by application.

A. By Model

Previous research has been conducted on PEA models separately, but much can be gained by studying them under a common viewpoint.

In Fig. 7, we provide a quick overview of different PEAs to point out important milestones in parallel computing with EAs. These "implementations" have rarely been studied as "parallel models." Instead, usually only the implementation itself is evaluated.

Some coarse-grain algorithms like dGA [126], DGENESIS [85], GALOPPS [53], PARAGENESIS [118], and PGA 2.5[11] are relatively close to the general model of migration islands. They often include many features to improve efficiency. Some other coarse-grain models like CoPDEB [2] and GDGA [65] have been designed for specific goals, such as providing explicit exploration/exploitation by applying different operators on each island. Another recent example of this class is the (iiGA) [78], which promotes coding and operator heterogeneity (see Section IV). A further parallel environment that offers adaptation with respect to the dynamic behavior of the computer pool and fault tolerance is MARS, described by Talbi *et al.* in [124].

Some other PGAs execute nonorthodox models of coarse-grain evolution. This is the case of GAMAS [98], based on using different alphabets in every island, and GENITOR II [135], based on a steady-state reproduction.

On the other hand, massive PEAs have been strongly associated to the machines on which they run: ASPARAGOS [55] and ECO-GA [34]. This is also the case of models of difficult classification (although most of the mentioned ones are of difficult classification!) like PEGAsuS [102], SP1-GA [75], or SGA-Cube [40]. As to the global parallelization model, some implementations, such as EnGENEer [104] or PGAPack [76], are available.

Finally, some efforts to construct general frameworks for PEAs are GAME [118], PEGAsuS, and RPL2 [101]. The

[11][Online] Available: http://www.aic.nrl.navy.mil/galist/src/pga-2.5.tar.Z

Parallel GA	Kind of Parallelism	Topology	Some Applications
ASPARAGOS	Fine grain. Applies hill-climbing if no improvement	Ladder	TSP
CoPDEB	Coarse grain. Every sub-pop. applies different operators	Fully Connected	Func. Opt. and ANN's
dGA	Distributed populations. Studies migration rate and freq.	Ring	Function Optimization
DGENESIS 1.0	Coarse grain with migrations among sub-populations	Any Desired	Function Optimization
ECO-GA	Fine grain. One of the first of its class	Grid	Function Optimization
EnGENEer	Global parallelization (parallel evaluations)	Master / Slave	Various
GALOPPS 3.1	Coarse grain. A very portable software	Any Desired	F. Opt. and Transport
GAMAS	Coarse grain. Uses 4 species of strings (nodes)	Fixed Hierarchy	ANN, Func. Opt., ...
GAME	Object oriented set of general programming tools	Any Desired	TSP, Func. Opt., ...
GDGA	Coarse Grain. Admits explicit exploration/exploitation	Hypercube	Func. Opt. (floating p.)
GENITOR II	Coarse grain. Interesting crossover operator	Ring	Func. Opt. and ANN's
HSDGA	Hierarchical coarse and fine grain GA. Uses E.S.	Ring, Tree, Star, ...	Function Optimization
iiGA	Injection island GA, heterogeneous and asynchronous	Node Hierarchy	Function Optimization
PARAGENESIS	Coarse grain. Made for the CM-200 (1 ind.⇔1 CPU)	Multiple	Function Optimization
PeGAsuS	Coarse or fine grain. High-level programming on MIMD	Multiple	Teaching and Func. Opt.
PGA	Sub-populations, migrate the best, local hill-climbing,...	Circular 2D Ladder	Func. Opt. and TSP
PGAPack	Global parallelization (parallel evaluations)	Master / Slave	Function Optimization
RPL2	Coarse grain. Very flexible to define new GA models	Any Desired	Research and Business
SGA-Cube	Coarse Grain. Implemented on the nCUBE 2	Hypercube	Function Optimization
SP1-GA	128 steady-state islands on an IBM SP1 with 128 nodes	2D Toroidal Mesh	Function Optimization

Fig. 7. A quick survey of the features of several parallel EAs.

Application Oriented	Algorithm Oriented		Tool Kits	
	Algorithm Specific	Algorithm Libraries	Educational	General Purpose
EVOLVER 2.0	ASPARAGOS	EM 2.1	GA Workbench	EnGENEer
OMEGA	CoPDEB	ESC^{A}P$_A$DE 1.2		GAGS
PC/BEAGLE	DGENESIS 1.0	GAlib		GAME
XpertRule GenAsys	ECO-GA	GENITOR I - II		PeGAsuS
	GAGA	LibGA		RPL2
	GALOPPS 3.1	OOGA		Splicer 1.0
	GAMAS	PGA 2.5		TOLKIEN
	GAucsd 1.2 / 1.4	PGAPack		
	GDGA	SUGAL 2.0		
	GENESIS 5.0 - GENEsYs 1.0			
	HSDGA			
	PARAGENESIS			
	SGA			
	SGA-Cube			

Fig. 8. Classification of EA implementations by type.

mentioned systems are endowed with "general" programming structures intended to ease the implementation of any model of PEA. The user must particularize these general structures to define his own algorithm. The result is sometimes called an *algorithmic skeleton*. Nowadays, many researchers are using *object-oriented programming* (OOP) to create better software for PEAs, but unfortunately some of the most important issues typical in OOP (such as making meaningful designs) are continuously being ignored in the resulting implementations. The reader can find some general guidelines for designing object-oriented PEAs in [9].

All these models and implementations offer different levels of flexibility, ranging from a single PEA to the specification of general PEA models. This list is not complete, of course, but it helps in describing the current "state of the art."

B. By Type

In order to complete our review, we now provide an extensive classification of sequential EAs and PEAs into three major categories [102], [5], according to their specific objectives (Fig. 8).

1) *Application Oriented:* These are black-box systems designed to hide the details of EAs and help the user in developing applications for specific domains. Some of these are useful for different purposes, such as scheduling or telecommunications (e.g., PC/BEAGLE). Others are much more application oriented (like OMEGA for finance). Usually, they are menu-driven and easily parameterizable.
2) *Algorithm Oriented:* Based on specific algorithms. The source code is available in order to provide an easy incorporation into new applications. This class may be further subdivided into the following.
   a) *Algorithm Specific:* They contain one single EA (e.g., GENESIS). Their users are system developers (for making applications) or EA researchers (interested in extensions).
   b) *Algorithm Libraries:* They support a group of algorithms in a library format (e.g., OOGA). They are highly parameterized and contain many different operators to help future applications.

3) *Tool Kits:* These are flexible environments for programming a range of different EAs and applications. They can be subdivided into the following.
    a) *Educational:* Used for introducing EA concepts to novice users (e.g., GA Workbench). The basic techniques to track executions and results during the evolution are easily managed.
    b) *General Purpose:* Useful for modifying, developing, and supervising a wide range of operators, algorithms and applications (e.g., Splicer).

Deeper explanations of these systems, detailed references, and up-to-date Internet pointers can be found in [5].

C. By Application

Parallel EAs and dEAs have been shown to be useful in practice in a number of industrial and commercial applications. Many real-life problems may need days or weeks of computing time to solve on serial machines. Although the intrinsic time complexity of a problem cannot be lowered by using a finite amount of computing resources in parallel, parallelism often allows reducing these times to reasonable levels. This can be very important in an industrial or commercial setting where the time to solution is instrumental for decision making and competitivity. Furthermore, the new models offered by structured populations often allow better exploration of alternative solutions of the design space.

PEAs have been used successfully in operations research (OR), engineering, and manufacturing, finance, VLSI, and telecommunication problems, among others. It is impossible to review such work in a small space, since in fact this would need a separate survey to deal with the so wide spectrum of optimization tasks in which PEAs are being used with success. However, for the sake of completeness, we do provide a description of a few successful and real-world important applications in very relevant fields of the optimization with PEAs. The references therein are intended to help the reader find more information on the subject.

Combinatorial optimization and OR: Many important combinatorial optimization problems, according to present knowledge, do not admit efficient, polynomial-time deterministic algorithms. It is thus worth looking for heuristic algorithms that are able to find at least good, if not optimal, solutions, but without any guarantee. EAs are heuristics that have been much used in the field of combinatorial optimization, usually with very satisfactory results. Allowing PEAs does not change the intrinsic complexity of the problem, but can be a big help both for reducing the execution time as well as to better explore the search space. There exist several studies dealing with parallel and dEAs in combinatorial optimization. Here, we mention the articles of Gorges–Schleuter [57] and Mühlenbein [89], in which large versions of the TSP and other prototypical problems in combinatorial optimization were solved satisfactorily. Parallel ESs have also been used for combinatorial optimization (see, for instance, [41] and [111]). It was shown by the authors that various forms of PEAs are competitive with other methods and sometimes even superior.

Telecommunication Network Design: In mobile telecommunication network design, two major problems have to be solved: the placement of the antennas and the frequency assignment. Both are hard multiobjective optimization problems and both have been satisfactorily solved with PEAs. The frequency assignment problem (FAP) consists in attributing frequencies to a number of radio links in such a way as to simultaneously satisfy a large number of constraints and use as few distinct frequencies as possible. Meunier *et al.* have recently used a parallel multiobjective GA for the FAP [87]. Since the fitness function is very expensive to evaluate in this problem and has a distinct spatial structure, its calculation was parallelized in a master–slave manner with PVM, with each worker taking care of a portion of a geographical area. The algorithm has been applied to real cases with good results. Also, we can find in [82] a good work in solving different instances of several FAP benchmarks by applying ant colony optimization algorithms (ACO). On the other side, see [20] for learning about a PEA used for radio network design (placement of antennas).

Financial Applications: Financial markets are inherently unpredictable since, according to a widely held view, the price time series essentially follows a random walk. However, more recent work has convincingly shown that although the broad picture is correct, there are still some significant deviations from a strict random process that can be made use of (see, for example [80]). Some of the technologies aiming at market forecasting make use of so-called *indicators*, which are simply elaborate statistics computed from observed prices. Once suitable indicators have been computed, they can be combined according to logical rules to build *trading models*, which are combinations of indicators and trading rules that can be used to generate trading recommendations. However, indicator computation is an extremely lengthy process and choosing between alternative trading models is a hard problem. Chopard *et al.* [28] used parallel multipopulation genetic programming techniques to speed up the search for good trading models with good results. The trading models induced with parallel GP are robust, yield good out-of-sample performance, and are produced in hours instead of days or weeks.

Design of Analog Electronic Circuits: Modern circuit design is a difficult task that poses a number of challenges to engineers. While considerable progress has been made in automating the design in the case of digital circuits, analog and analog-digital circuit design has not enjoyed similar developments and somehow remains a form of art rather than solid engineering. By using heavy-duty parallel genetic programming, Koza and coworkers [73] have been able to synthesize complex analog circuits automatically from a high-level description of the circuit's desired function. Genetic programming has produced designs that are competitive with human-designed circuits, and even better in some cases. Furthermore, the GP approach is not limited to particular cases; it can be generalized directly to other problems. Parallelism has been instrumental in these calculations since they are extremely time-consuming.

VLSI Design: VLSI routing and placement problems are very important industrial problems. They are notoriously difficult to solve with standard algorithms due to the exponential time complexity increase with the instance size. Heuristic and approximate algorithms are thus the only practical way of obtaining good solutions in reasonable times. An interesting application of parallel GAs to optimize routing in VLSI circuits subject to other industrially relevant constraints is presented in [77]. The author uses a classical island model with grid communication topology among subpopulations and also examines several policies for the selection of migrants and the migration frequency. The results are at least as good as those obtained with other popular methodologies, including sequential EAs, on a number of standard benchmark problems in the field.

Engineering Design: Good examples of parallel GAs in engineering design can be found, for example, in the work of Doorly and his group on aeronautical design optimization [36]. For another successful application, in [39] the authors combined parallel GAs with finite element methods for the optimization of flywheels. They compare a number of optimization methods including a standard island-parallel GA and the iiGA [78], which has been described in Section IV. In [39], the authors show that the iiGA is the most effective for the flywheel optimization problem and offers a number of distinct advantages over standard PEAs and over other commonly used methods such as simulated annealing. Again, the time to solution using PEAs allows one to either afford a better solution quality for a given problem size in less time, or a solution to a more realistic and bigger problem instance in the same time, which are both valuable improvements in engineering practice.

X. Summary

This paper contains a modern survey of parallel models and implementations of EAs. We have stressed not only the associated algorithmic issues, but also the parallel tools for building PEAs. By summarizing the parallel algorithms, their applications, classes, and theoretical foundations, we intend to offer valuable information not only for beginners, but also for researchers working with EAs or heuristics in general.

The list of references has been elaborated to serve as a directory for granting the reader access to the valuable results that PEAs are offering to the research community. Most important trends have been discussed, yielding what we hope is a unified overview and a useful text.

Acknowledgment

The authors gratefully acknowledge the comments of the reviewers that helped in improving the contents of this work notably.

References

[1] P. Adamidis. (1994, Nov.) Review of parallel genetic algorithms nibliography. Tech. Rep., Aristotle Univ., Thessaloniki, Greece. [Online]. Available: http://www.control.ee.auth.gr/panos/papers/p̃ga_review.ps.gz

[2] P. Adamidis and V. Petridis, "Co-operating populations with different evolution behavior," in *Proc. 1996 IEEE Conf. Evolutionary Computation*. Piscataway, NJ: IEEE Press, 1996, pp. 188–191.

[3] S. G. Akl, *The Design and Analysis of Parallel Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[4] E. Alba, "Parallel evolutionary algorithms can achieve superlinear performance," *Inform. Process. Lett.*, vol. 82, no. 1, pp. 7–13, Apr. 2002.

[5] E. Alba and J. M. Troya, "A survey of parallel distributed genetic algorithms," *Complexity*, vol. 4, no. 4, pp. 31–52, 1999.

[6] ——, "Cellular evolutionary algorithms: Evaluating the influence of ratio," in *Parallel Problem Solving from Nature, PPSN VI*. ser. Lecture Notes in Computer Science, M. Schoenauer *et al.*, Eds. New York: Springer-Verlag, 2000, vol. 1917, pp. 29–38.

[7] ——, "Influence of the migration policy in parallel distributed gas with structured and panmictic populations," *Appl. Intell.*, vol. 12, no. 3, pp. 163–181, 2000.

[8] ——, "Analyzing synchronous and asynchronous parallel distributed genetic algorithms," *Future Gener. Comput. Syst.*, vol. 17, pp. 451–465, Jan. 2001.

[9] ——, "Gaining new fields of application for oop: The parallel evolutionary algorithm case," *J. Object Oriented Programming*, Dec. 2001.

[10] ——, "Improving flexibility and efficiency by adding parallelism to genetic algorithms," *Statist. Comput.*, vol. 12, no. 2, pp. 91–114, 2002.

[11] D. Andre and J. R. Koza, "Parallel genetic programming: A scalable implementation using the transputer network architecture," in *Advances in Genetic Programming 2*, P. Angeline and K. Kinnear, Eds. Cambridge, MA: MIT Press, 1996, pp. 317–337.

[12] ——, "A parallel implementation of genetic programming that achieves super-linear performance," *J. Inform. Sci.*, vol. 106, no. 3–4, pp. 201–218, 1998.

[13] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. Oxford, U.K.: Oxford Univ. Press, 1996.

[14] *Handbook of Evolutionary Computation*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., Oxford Univ. Press, Oxford, U.K., 1997.

[15] *Handbook of Evolutionary Computation*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., Oxford Univ. Press, Oxford, U.K., 1997, pp. C3-2:1–C3-2:14.

[16] S. Baluja, "Structure and performance of fine-grain parallelism in genetic search," in *Proc. 5th Int. Conf. Genetic Algorithms*, S. Forrest, Ed., 1993, pp. 155–162.

[17] T. C. Belding, "The distributed genetic algorithm revisited," in *Proc. 6th Int. Conf. Genetic Algorithms*, L. J. Eshelman, Ed., 1995, pp. 114–121.

[18] S. Benkner, "VFC: The Vienna fortran compiler," *Sci. Program.*, vol. 7, no. 1, pp. 67–81, 1999.

[19] W. Bossert, "Mathematical optimization: Are there abstract limits on natural selection?," in *Mathematical Challenges to the Neo-Darwinian Interpretation of Evolution*, P. S. Moorhead and M. M. Kaplan, Eds. Philadelphia, PA: Wistar Inst. Press, 1967, pp. 35–46.

[20] P. Calégari, F. Guidec, P. Kuonen, and D. Kobler, "Parallel island-based genetic algorithm for radio network design," *J. Parallel Distrib. Comput.*, vol. 47, pp. 86–90, 1997.

[21] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs Parallèles, Réseaux et Systèmes Répartis*, vol. 10, no. 2, pp. 141–171, 1998.

[22] ——, *Efficient and Accurate Parallel Genetic Algorithms*. Norwell, MA: Kluwer, 2000.

[23] E. Cantú-Paz and D. E. Goldberg, "Predicting speedups of idealized bounding cases of parallel genetic algorithms," in *Proc. 7th Int. Conf. Genetic Algorithms*, T. Bäck, Ed., 1997, pp. 113–120.

[24] ——, "On the scalability of parallel genetic algorithms," *Evolut. Comput.*, vol. 7, no. 4, pp. 429–449, 1999.

[25] M. Capcarrère, M. Tomassini, A. Tettamanzi, and M. Sipper, "A statistical study of a class of cellular evolutionary algorithms," *Evolut. Comput.*, vol. 7, no. 3, pp. 255–274, 1999.

[26] A. Chipperfield and P. Fleming, *Parallel and Distributed Computing Handbook—Parallel Genetic Algorithms*, A. Y. H. Zomaya, Ed. New York: McGraw-Hill, 1996, pp. 1118–1143. .

[27] F. S. Chong, "A Java based distributed approach to genetic programming on the Internet," Master's thesis, Univ. Birmingham, Birmingham, AL, 1998.

[28] B. Chopard, O. Pictet, and M. Tomassini, "Parallel and distributed evolutionary computation for financial applications," *Parallel Algorith. Applic.*, vol. 15, pp. 15–36, 2000.

[29] C. A. Coello, "A comprehensive survey of evolutionary-based multiobjective optimization techniques," *Knowledge Inform. Syst.*, vol. 1, no. 3, pp. 269–308, Aug. 1999.

[30] J. P. Cohoon, S. U. Hegde, W. N. Martin, and D. Richards, "Punctuated equilibria: A parallel genetic algorithm," in *Proc. 2nd Int. Conf. Genetic Algorithms*, J. J. Grefenstette, Ed., 1987, pp. 148–154.

[31] D. E. Comer and D. L. Stevens, *Internetworking with TCP/IP (Volume III)*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[32] C. Cotta, E. Alba, and J. M. Troya, "Stochastic reverse hillclimbing and iterated local search," in *Proc. 1999 Congress Evolutionary Computation*. Piscataway, NJ: IEEE Press, 1999, vol. 2, pp. 1558–1565.

[33] J. M. Daida, S. J. Ross, and B. C. Hannan, “Biological symbiosis as a metaphor for computational hybridization.,” in *Proc. 6th Int. Conf. Genetic Algorithms*, L. J. Eshelman, Ed., 1995, pp. 328–335.

[34] Y. Davidor, “A naturally occurring niche and species phenomenon: The model and first results,” in *Proc. 4th Int. Conf. Genetic Algorithms*, R. K. Belew and L. B. Booker, Eds., 1991, pp. 257–263.

[35] *Handbook of Evolutionary Computation—Specification Methods*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., Oxford Univ. Press, Oxford, U.K., 1997, pp. C6.2:1–C6.2:5.

[36] D. J. Doorly, J. Peiró, T. Kuan, and J.-P. Oesterle, “Optimization of airfoils using parallel genetic algorithms,” in *Proc. 15th AIAA Int. Conf. Numerical Methods in Fluid Mechanics*, Monterey, CA, 1996.

[37] S. Droste and D. Wiesmann, “On Representation and Genetic Operators in Evolutionary Algorithms,” Univ. Dortmund, FB Informatik, SFB 531, Dortmund, Germany, Tech. Rep. CI 41/98, 1998.

[38] B. S. Duncan, “Parallel evolutionary programming,” in *Proc. 2nd Annu. Conf. Evolutionary Programming*, D. B. Fogel and W. Ahmar, Eds. La Jolla, CA: Evolut. Prog. Soc., 1993, pp. 202–208.

[39] D. Eby, R. Averill, W. Punch, and E. D. Goodman, “Optimal design of flywheels using an injection island genetic algorithm,” *Artif. Intell. Eng., Des., Manuf.*, vol. 13, pp. 289–402, 1999.

[40] J. A. Erickson, R. E. Smith, and D. E. Goldberg, “SGA-Cube, a Simple Genetic Algorithm for Ncube 2 Hypercube Parallel Computers,” Univ. Alabama, Birmingham, Tech. Rep. 91 005, 1991.

[41] I. De Falco, R. Del Balio, E. Tarantino, and R. Vaccaro, “Testing parallel evolution strategies on the quadratic assignment problem,” *Proc. IEEE 1993 Int. Conf. Systems, Man and Cybernetics*, vol. V, pp. 254–259, 1993.

[42] F. Fernández, M. Tomassini, W. F. Punch III, and J. M. Sánchez, “Experimental study of multipopulation parallel genetic programming,” in *Proc. EuroGP’2000—Genetic Programming*, vol. 1802, R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, Eds., 2000, pp. 283–293.

[43] F. Fernández, M. Tomassini, and L. Vanneschi, “Studying the influence of communication topology and migration on distributed genetic programming,” in *Proc. EuroGP’2001—Genetic Programming*, vol. 2038, J. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. Tettamanzi, and W. B. Langdon, Eds., 2001, pp. 50–63.

[44] D. B. Fogel and A. Ghozeil, “A note on representations and variation operators,” *IEEE Trans. Evol. Comput.*, vol. 1, pp. 159–161, July 1997.

[45] Message Passing Interface Forum, “MPI: A message-passing interface standard,” *Int. J. Supercomput. Applic.*, vol. 8, no. 3–4, pp. 165–414, 1994.

[46] I. Foster and C. Kesselmann, “Globus: A metacomputing infrastructure toolkit,” *J. Supercomput. Applic.*, vol. 11, no. 2, pp. 115–128, 1997.

[47] ——, *The Grid: Blueprint for a New Computing Infrastructure*. San Mateo, CA: Morgan Kaufmann, 1999.

[48] F. Glover, “A user’s guide to tabu search,” *Annals Oper. Res.*, vol. 41, pp. 3–28, 1993.

[49] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[50] ——, “Sizing populations for serial and parallel genetic algorithms,” in *Proc. 3rd Int. Conf. Genetic Algorithms*, J. D. Schaffer, Ed., 1989, pp. 70–79.

[51] D. E. Goldberg and K. Deb, “A comparative analysis of selection schemes used in genetic algorithms,” in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Mateo, CA: Morgan Kaufmann, 1991, pp. 69–93.

[52] D. E. Goldberg and R. E. Smith, “Nonstationary function optimization using genetic algorithms with dominance and diploidy,” in *Proc. 2nd Int. Conf. Genetic Algorithms*, J. J. Grefenstette, Ed., 1987, pp. 59–68.

[53] E. D. Goodman, “An Introduction to GALOPPS v3.2,” GARAGe, I. S. Lab., Dpt. of C. S. and C.C.C.A.E.M., Michigan State Univ., East Lansing, MI, Tech. Rep. 96-07-01, 1996.

[54] V. S. Gordon and D. Whitley, “Serial and parallel genetic algorithms as function optimizers,” in *Proc. 5th Int. Conf. Genetic Algorithms*, S. Forrest, Ed., 1993, pp. 177–183.

[55] M. Gorges-Schleuter, “ASPARAGOS an asynchronous parallel genetic optimization strategy,” in *Proc. 3rd Int. Conf. Genetic Algorithms*, J. D. Schaffer, Ed., 1989, pp. 422–427.

[56] ——, “Asparagos96 and the traveling salesman problem,” in *Proc. 4th Int. Conf. Evolutionary Computation*, T. Bäck, Ed., 1997, pp. 171–174.

[57] ——, “On the power of evolutionary optimization at the example of ATSP and large TSP problems,” in *Proc. 4th Eur. Conf. Artificial Life*, P. Husbands and I. Harvey, Eds., 1997.

[58] ——, “An analysis of local selection in evolution strategies,” in *Proc. Genetic and Evolutionary Conf. GECCO99*, vol. 1, 1999, pp. 847–854.

[59] J. J. Grefenstette, “Parallel adaptive algorithms for function optimization,” Vanderbilt Univ., Nashville, TN, Tech. Rep. CS-81-19, 1981.

[60] A. S. Grimshaw and W. A. Wulf, “The legion vision of a worldwide virtual computer,” *Commun. ACM*, vol. 40, no. 1, pp. 39–45, 1997.

[61] P. B. Grosso, “Computer simulation of genetic adaptation: Parallel subcomponent interaction in a multilocus model,” Ph.D. dissertation, Univ. Michigan, Ann Arbor, 1985.

[62] UEA CALMA Group, “Calma project report 2.4: Parallelism in combinatorial optimization,” School of Inform. Syst., Univ. East Anglia, Norwich, U.K., Tech. Rep., Sept. 18, 1995.

[63] F. Gruau, “Neural networks synthesis using cellular encoding and the genetic algorithm,” Ph.D. dissertation, Univ. Claude Bernard-Lyon I, Lyon, France, 1994.

[64] W. E. Hart, S. Baden, R. K. Belew, and S. Kohn, “Analysis of the numerical effects of parallelism on a parallel genetic algorithm,” in IEEE Proc. Workshop on Solving Combinatorial Optimization Problems in Parallel (IPPS97), [CD-ROM], IEEE, Piscataway, NJ, 1997.

[65] F. Herrera and M. Lozano, “Gradual distributed real-coded genetic algorithms,” *IEEE Trans. Evol. Comput.*, vol. 4, pp. 43–63, 2000.

[66] F. Herrera, M. Lozano, and C. Moraga, “Hybrid distributed real-coded genetic algorithms,” in *Parallel Problem Solving from Nature, PPSN IV*. ser. Lecture Notes in Computer Science, A. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, Eds. Berlin, Germany: Springer-Verlag, 1998, vol. 1498, pp. 603–612.

[67] W. D. Hills, “Co-evolving parasites improve simulated evolution as an optimization procedure.,” in *Artificial Life II*. ser. Studies in the Sciences of Complexity, C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, Eds. Redwood City, CA: Addison-Wesley, 1992, vol. X, pp. 313–324.

[68] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michigan Press, 1975.

[69] K. Hwang, *Advanced Computer Architecture*. New York: McGraw Hill, 1993.

[70] JavaSoft. RMI: The JDK 1.1 Specification (1997). [Online]. Available: javasoft.com/products/jdk/1.1/docs/guide/rmi/index.html

[71] K. A. De Jong, “An Analysis of the Behavior of a Class of Genetic Adaptive Systems,” Ph.D. Thesis, Univ. Michigan, 1975. 5140B, university microfilms no. 76-9381.

[72] S. Khuri, T. Bäck, and J. Heitkötter, “An evolutionary approach to combinatorial optimization problems,” in *Proc. 22nd ACM Computer Science Conf.*, 1994, pp. 66–73.

[73] J. Koza, F. H. Bennet III, D. Andre, and M. A. Keane, “The design of analog circuits by means of genetic programming,” in *Evolutionary Design by Computers*, P. J. Bentley, Ed. San Francisco, CA: Morgan Kaufmann, 1999, pp. 365–385.

[74] P. J. Van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*. Norwell, MA: Kluwer, 1988.

[75] D. Levine, “A Parallel Genetic Algorithm for the Set Partitioning Problem,” Argonne Nat. Lab., Math. Comput. Sci. Div., Argonne, France, Tech. Rep. ANL-94/23, 1994.

[76] ——, “Users Guide to the PGAPack Parallel Genetic Algorithm Library,” Argonne Nat. Lab. , Math. Comput. Sci. Div., Tech. Rep. ANL-95/18, Jan. 31, 1995.

[77] J. Lienig, “A parallel genetic algorithm for performance-driven VLSI routing,” *IEEE Trans. Evol. Comput.*, vol. 1, pp. 29–39, Apr. 1997.

[78] S. C. Lin, W. F. Punch, and E. D. Goodman, “Coarse-grain parallel genetic algorithms: Categorization and a new approach,” in *Proc. 6th IEEE SPDP*, 1994, pp. 28–37.

[79] S. H. Lin, E. D. Goodman, and W. F. Punch, “Investigating parallel genetic algorithms on job shop scheduling problems.,” in *Proc. 6th Int. Conf. Evolutionary Programming*, P. Angeline, R. Reynolds, J. McDonnell, and R. Eberhart, Eds. Berlin, Germany: Springer, 1997, pp. 383–393.

[80] B. Mandelbrot, *Fractals and Scaling in Finance*. New York: Springer, 1997.

[81] B. Manderick and P. Spiessens, “Fine-grained parallel genetic algorithms,” in *Proc. 3rd Int. Conf. Genetic Algorithms*, J. D. Schaffer, Ed., 1989, pp. 428–433.

[82] V. Maniezzo and A. Carbonaro, “An ants heuristic for the frequency assignment problem,” *Future Gener. Comput. Syst.*, vol. 16, pp. 927–935, 2000.

[83] W. N. Martin, J. Liening, and J. P. Cohoon, “Island (migration) models: Evolutionary algorithms based on punctuated equilibria,” in *Evolutionary Computation 2: Advanced Algorithms and Operators*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds. London, U.K.: IOP Publishing, 2000, vol. 2, pp. 101–124.

[84] T. Maruyama, T. Hirose, and A. Konagaya, "A fine-grained parallel genetic algorithm for distributed parallel systems," in *Proc. 5th Int. Conf. Genetic Algorithms*, S. Forrest, Ed., 1993, pp. 184–190.

[85] M. Mejía-Olvera and E. Cantú-Paz, "DGENESIS-software for the execution of distributed genetic algorithms," in *Proc. XX Conf. Latinoamericana de Informática*, 1994, pp. 935–946.

[86] R. Menke, "A Revision of the Schema Theorem," Univ. Dortmund, FB Informatik, SFB 531, Dortmund, Germany, Tech. Rep. CI 14/97, Dec. 4, 1997.

[87] H. Meunier, E. G. Talbi, and P. Reininger, "A multiobjective genetic algorithm for radio network optimization," in *Proc. Congress on Evolutionary Computation 2000*, 2000, pp. 317–324.

[88] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed, Berlin, Germany: Springer-Verlag, 1996.

[89] H. Mühlenbein, "Parallel genetic algorithms in combinatorial optimization," in *Computer Science and Operations Research*, O. Balci, R. Sharda, and S. Zenios, Eds. New York: Pergamon, 1992, pp. 441–456.

[90] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm I. Continuous parameter optimization," *Evolut. Comput.*, vol. 1, no. 1, pp. 25–49, 1993.

[91] H. Mühlenbein, M. Schomisch, and J. Born, "The parallel genetic algorithm as function optimizer," *Parallel Comput.*, vol. 17, pp. 619–632, Sept. 1991.

[92] M. Munetomo, Y. Takai, and Y. Sato, "An efficient migration scheme for subpopulation-based ansynchronously parallel genetic algorithms," in *Proc. 5th Int. Conf. Genetic Algorithms*, S. Forrest, Ed., 1993, p. 649.

[93] B. Naudts, D. Suys, and A. Verschoren, "Epistasis as a basic concept in formal landscape analysis," in *Proc. 7th Int. Conf. Genetic Algorithms*, T. Bäck, Ed., 1997, pp. 65–72.

[94] K. M. Nelson, "Function optimization and parallel evolutionary programming on the MasPar MP-1," in *Proc. 3rd Annu. Conf. Evolutionary Programming*, A. V. Sebald and L. J. Fogel, Eds., 1994, pp. 324–334.

[95] T. Niwa and M. Tanaka, "Analysis on the island model parallel genetic algorithms for the genetic drifts," in *Proc. Simulated Evolution and Learning (SEAL'98)*, B. McKay, X. Yao, C. S. Newton, J. H. Kim, and T. Furuhashi, Eds. River Edge, NJ: World Scientific, 1998, pp. 349–356.

[96] M. Oussaidène, B. Chopard, O. Pictet, and M. Tomassini, "Parallel genetic programming and its application to trading model induction," *Parallel Comput.*, vol. 23, pp. 1183–1198, 1997.

[97] C. C. Pettey and M. R. Leuze, "A theoretical investigation of a parallel genetic algorithm," in *Proc. 3rd Int. Conf. Genetic Algorithms*, J. D. Schaffer, Ed., 1989, pp. 398–405.

[98] J. C. Potts, T. D. Giddens, and S. B. Yadav, "The development and evaluation of an improved genetic algorithm based on migration and artificial selection," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 73–86, 1994.

[99] W. Punch, "How effective are multiple populations in genetic programming," in *Proc. 3rd Annu. Conf. Genetic Programming 1998*, J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, D. Goldberg, H. Iba, and R. L. Riolo, Eds., 1998, pp. 308–313.

[100] N. J. Radcliffe and P. D. Surry, "Fitness variance of formae and performance prediction," in *Foundations of Genetic Algorithms*, L. D. Whitley and M. D. Vose, Eds. San Mateo, CA: Morgan Kaufmann, 1995, vol. 3, pp. 51–72.

[101] ——, "The reproductive plan language RPL2: Motivation, architecture and applications," in *Genetic Algorithms in Optimization, Simulation and Modeling*, J. Stender, E. Hillebrand, and J. Kingdon, Eds. Amsterdam, The Netherlands: IOS Press, 1999.

[102] J. L. Ribeiro-Filho, C. Alippi, and P. Treleaven, "Genetic algorithm programming environments," in *Parallel Genetic Algorithms: Theory and Applications*, J. Stender, Ed. Amsterdam, The Netherlands: IOS Press, 1993, pp. 65–83.

[103] G. A. Riessen, J. Williams, and X. Yao, "PEPNet: Parallel evolutionary programming for constructing artificial neural networks," in *Proc. 6th Int. Conf. Evolutionary Programming*, P. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, Eds, Berlin, Germany: Springer, 1997, vol. 1213, pp. 35–45.

[104] G. Robbins, "EnGENEer—The evolution of solutions," in *Proc. 5th Annu. Seminar Neural Networks and Genetic Algorithms*, London, U.K., 1992.

[105] G. Rudolph, "Global optimization by means of distributed evolution strategies," in *Parallel Problem Solving from Nature*, vol. 496, H.-P. Schwefel and R. Männer, Eds., 1991, pp. 209–213.

[106] G. Rudolph and J. Sprave, "A cellular genetic algorithm with self-adjusting acceptance thershold," in *Proc. 1st IEE/IEEE Int. Conf. Genetic Algorithms in Engineering Systems: Innovations and Applications*, 1995, pp. 365–372.

[107] ——, "Significance of locality and selection pressure in the Grand Deluge evolutionary algorithm," in *Parallel Problem Solving from Nature—PPSN IV*, vol. 1141, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds., 1996, pp. 666–675.

[108] J. Sarma and K. A. De Jong, "An analysis of the effect of the neighborhood size and shape on local selection algorithms," in *Parallel Problem Solving from Nature (PPSN IV)*, ser. Lecture Notes in Computer Science, H. M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Eds., 1996, vol. 1141, pp. 236–244.

[109] ——, "An analysis of local selection algorithms in a spatially structured evolutionary algorithm," in *Proc. 7th Int. Conf. Genetic Algorithms*, T. Bäck, Ed., 1997, pp. 181–186.

[110] ——, "The behavior of spatially distributed evolutionary algorithms in nonstationary environments," in *Proc. Genetic Evolutionary Computation Conf.*, W. Banzhaf *et al.*, Eds., 1999, pp. 572–578.

[111] M. Schütz and J. Sprave, "Application of parallel mixed-integer evolution strategies with mutation rate pooling.," in *Evolutionary Programming V—Proc. 5th Annu. Conf. Evolutionary Programming (EP'96)*, L. J. Fogel, P. J. Angeline, and T. Báck, Eds., 1996, pp. 345–354.

[112] M. Sefrioui and J. Périaux, "A hierarchical genetic algorithm using multiple models for optimization," in *Parallel Problem Solving from Nature—PPSN VI*. ser. Lecture Notes in Computer Science, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, Eds, Berlin: Springer-Verlag, 2000, vol. 1917, pp. 879–888.

[113] R. Shonkwiler, "Parallel genetic algorithms," in *Proc. 5th Int. Conf. Genetic Algorithms*, S. Forrest, Ed., 1993, pp. 199–205.

[114] M. Sipper, *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*, Berlin, Germany: Springer-Verlag, 1997.

[115] G. Spezzano, G. Folino, and C. Pizzuti, "CAGE: A tool for parallel genetic programming applications," in *Proc. EuroGP'2001 Genetic Programming*, vol. , 2038, J. Miller, M. Tomassini, P. L. Lanzi, C. Ryan, A. Tettamanzi, and W. B. Langdon, Eds., 2001, pp. 64–73.

[116] P. Spiessens and B. Manderick, "A massively parallel genetic algorithm," in *Proc. 4th Int. Conf. Genetic Algorithms*, L. B. Booker and R. K. Belew, Eds., 1991, pp. 279–286.

[117] J. Sprave, "A unified model of nonpanmictic population structures in evolutionary algorithms," in *Proc. Congress on Evolutionary Computation (CEC'99)*, NJ, 1999, pp. 1384–1391.

[118] J. Stender, Ed., *Parallel Genetic Algorithms: Theory and Applications*. Amsterdam, The Netherlands: IOS Press, 1993.

[119] V. S. Sunderam, "PVM: A framework for parallel distributed computing," *J. Concurr. Practice and Experience*, vol. 2, no. 4, pp. 315–339, 1990.

[120] V. S. Sunderam and G. A. Geist, "Heterogeneous parallel and distributed computing," *Parallel Comput.*, vol. 25, pp. 1699–1721, 1999.

[121] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*, J. D. Schaffer, Ed., 1989, pp. 2–9.

[122] ——, "A study of reproduction in generational and steady-state genetic algorithms," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed. San Mateo, CA: Morgan Kaufmann, 1991, pp. 94–101.

[123] E.-G. Talbi, "A Taxonomy of Hybrid Metaheuristics," Univ. Lille, Lab. Inform. Fundam., Lille, France, Tech. Rep. TR-183, May 1998.

[124] E.-G. Talbi, Z. Hafidi, D. Kebbal, and J.-M. Geib, "MARS: An adaptive parallel programming environment," in *High Performance Cluster Computing*, B. Rajkumar, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1999, vol. 1, pp. 722–739.

[125] R. Tanese, "Parallel genetic algorithms for a hypercube," in *Proc. 2nd Int. Conf. Genetic Algorithms*, J. J. Grefenstette, Ed., 1987, p. 177.

[126] ——, "Distributed genetic algorithms," in *ICGA-3*, J. D. Schaffer, Ed., 1989, pp. 434–439.

[127] A. Tettamanzi and M. Tomassini, *Soft Computing: Integrating Evolutionary, Neural and Fuzzy Systems*. New York: Springer, 2001.

[128] M. Tomassini, "The parallel genetic cellular automata: Application to global function optimization," in *Proc. Int. Conf. Artificial Neural Networks and Genetic Algorithms*, R. F. Albrecht, C. R. Reeves, and N. C. Steele, Eds. New York, 1993, pp. 385–391.

[129] A. Umar, *Object-Oriented Client/Server Internet Environments*. Englewood Cliffs, NJ: Prentice-Hall, 1997.

[130] J. Wakunda and A. Zell, "Median-selection for parallel steady-state evolution strategies," in *Proc. 6th Int. Conf. Parallel Problem Solving from Nature—PPSN VI*, vol. 1917, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, Eds., Sept. 16–20, 2000, pp. 405–414.

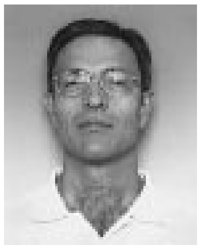
[131] D. Whitley, "Cellular genetic algorithms," in *Proc. 5th Int. Conf. Genetic Algorithms*, S. Forrest, Ed., 1993, p. 658.

[132] ——, "A genetic algorithm tutorial," *Statist. Comput.*, vol. 4, pp. 65–85, 1994.

[133] D. Whitley, S. Rana, J. Dzubera, and K. E. Mathias, "Evaluating evolutionary algorithms," *Artific. Intell.*, vol. 85, pp. 245–276, 1997.
[134] D. Whitley, S. Rana, and R. B. Heckendorn, "Island model genetic algorithms and linearly separable problems," in *Evolutionary Computing: Proc. AISB Workshop, Lecture Notes in Computer Science*, vol. 1305, D. Corne and J. L. Shapiro, Eds., 1997, pp. 109–125.
[135] D. Whitley and T. Starkweather, "GENITOR II: A distributed genetic algorithm," *J. Expt. Theor. Artif. Intell.*, vol. 2, pp. 189–214, 1990.

Enrique Alba received the Ph.D. degree in designing and analyzing parallel and distributed genetic algorithms.

He is an Associate Professor of Computer Science at the University of Málaga, Málaga, Spain. His current research interests involve the design and application of evolutionary algorithms, neural networks, and other bio-inspired systems to real problems including telecommunications, combinatorial optimization, and neural network design. The main focus of all his work is on parallelism. He has published many scientific papers in international conferences and journals, and holds national and international awards for his research results.



Marco Tomassini is a Professor of Computer Science at the University of Lausanne, Lausanne, Switzerland. His current research interests involve the application of biological principles to artificial systems, including evolutionary computation, evolvable hardware, complex adaptive systems, artificial life, fuzzy logic, and artificial neural networks. He is also interested in cognitive sciences, machine learning, parallel computation, cellular automata and on hybrid and cooperative methods for searching and optimization. He has been the Program Chairman of several international events and has published many scientific papers and several books in these fields.