Parallel Genetic Algorithms: A Useful Survey

TOMOHIRO HARADA, Ritsumeikan University, Japan
ENRIQUE ALBA, Universidad de Malága, Spain

In this article, we encompass an analysis of the recent advances in parallel genetic algorithms (PGAs). We have selected these algorithms because of the deep interest in many research fields for techniques that can face complex applications where running times and other computational resources are greedily consumed by present solvers, and PGAs act then as efficient procedures that fully use modern computational platforms at the same time that allow the resolution of cutting-edge open problems. We have faced this survey on PGAs with the aim of helping newcomers or busy researchers who want to have a wide vision on the field. Then, we discuss the most well-known models and their implementations from a recent (last six years) and useful point of view: We discuss on highly cited articles, keywords, the venues where they can be found, a very comprehensive (and new) taxonomy covering different research domains involved in PGAs, and a set of recent applications. We also introduce a new vision on open challenges and try to give hints that guide practitioners and specialized researchers. Our conclusion is that there are many advantages to using these techniques and lots of potential interactions to other evolutionary algorithms; as well, we contribute to creating a body of knowledge in PGAs by summarizing them in a structured way, so the reader can find this article useful for practical research, graduate teaching, and as a pedagogical guide to this exciting domain.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Mathematics of computing → Evolutionary algorithms**; • **Theory of computation → Evolutionary algorithms**; • **Computing methodologies → Parallel algorithms**; **Genetic algorithms**;

Additional Key Words and Phrases: Parallelism, genetic algorithms, complex problem solving, time consuming applications, new areas for search, optimization, and learning, last five years, artificial intelligence

ACM Reference format:
Tomohiro Harada and Enrique Alba. 2020. Parallel Genetic Algorithms: A Useful Survey. *ACM Comput. Surv.* 53, 4, Article 86 (August 2020), 39 pages.
https://doi.org/10.1145/3400031

1 INTRODUCTION

Let us start this article by answering the most important obvious question: Why should we bother reading about Parallel Genetic Algorithms (PGAs)? The answer is not as obvious as one would think ("because we can solve complex problems in a more efficient manner"), because it goes to the root of world research, to engineering, and to the importance of optimization in all this (as

This research has been partially funded by the Spanish MINECO and FEDER project TIN2017-88213-R (6city at http://6city.lcc.uma.es), EU H2020-ICT-2019-3 (TAILOR), and Andalusian/FEDER UMA18-FEDERJA-003 (PRECOG at http://precog.lcc.uma.es).
Authors' addresses: T. Harada (corresponding author), Ritsumeikan University, 1-1-1 Nojihigashi, Kusatsu, Shiga, 5258577, Japan; email: harada@ci.ritsumei.ac.jp; E. Alba, Universidad de Malága, Malága, Spain; email: eat@lcc.uma.es.

0360-0300/2020/08-ART86 $15.00
https://doi.org/10.1145/3400031

well as search and machine learning). If we focus on optimization, we soon will notice that most disciplines and works on apparently diverse fields finally rely on optimizing a complex system: from building new drugs [37, 47, 74] (optimize the energy estate of chemicals, like their angles), deploying better 5Gs networks [42] (optimized covered area, the packet protocol parameters, and the number of potential users), investing in the stock market [19, 102] (maximize revenues of a complex portfolio), or helping citizens to move into a city [100, 101, 108, 111] (minimize pollution, fuel consumption, travel time, maximize comfort and drivers' experience), just to name a few. Filling your fridge with healthy food at low cost, combat a fire, program your computer with a minimum number of bugs, and thousands of other applications are now possible by using GAs [4].

These *metaheuristics* (also known as *bio-inspired techniques*) [20, 70, 105] can address applications where the target goal function is non-derivable, non-continuous, ill-defined, uses arbitrary mixed types of variables, has many competing goals, or even does not have any analytical expression at all. This wide applicability made GAs to penetrate most domains of research, development, and innovation [5]. The massive benefits of these techniques took them to multidisciplinary applications and starred lots of success stories in their short-term (30 years) existence. And, paired with this impressive presentation card, the main problem always arrives at practitioners and researchers: Studies and products based in GAs need long-running times as soon as they start being useful. These long-running times become relevant for many reasons: (a) a large dimensionality of the problem, (b) a need to use customized operators for the application, (c) a complex target function that takes several minutes of computation, (d) a need of dealing with large datasets in the algorithm, (e) a high number of non-linear restrictions . . . and still for some other reasons not shown here. When researchers arrive at this *wall*, parallelism is one way to jump over it for improved performance.

But long-running times are just one way to arrive at PGAs. In fact, it was quite a while since researchers have shown that PGAs are not just faster algorithms. They are much more than this; they are new models of research that can profit from parallel platforms such as clusters, multiprocessors, GPUs, and others [3, 107]. Indeed, an additional source for speedup (time gains) is the potential creation of new techniques that search, optimize, and learn in a numerically faster way, by saving function calls (the usual way of measuring numerical efficiency). Thus, much can be gained by, e.g., running a distributed algorithm on a network [12, 45, 119] or a cellular model on a GPU [115]. This separation between model and implementation is worth knowing, and we will organize one of our sections according to this.

In facing this survey, we had to make decisions. First, we decided not to go for evolutionary algorithms (EAs) in general, since the topic of PGAs will fade away. In fact, GAs are maybe the most popular class of EAs, and most things done on them can be reproduced in other EAs and population-based techniques, so it seems worthy to focus-to-tell, instead of making a too abstract survey. Second, we decided to focus on analyzing the works published in the period of the recent six years from 2013 to 2018. If the search period is too broad, the papers needed to be analyzed will be enormous, which makes it difficult to discuss the latest advances in depth. Moreover, there are other surveys regarding PGAs and PEAs, for example, by Alba et al. in 2013 [7] or by Talbi in 2015 [106], that readers can check out for older works in this field. We here add new ways of surveying by (1) making it thorough and structured, (2) introducing technological discussions, (3) considering multidisciplinary sources of information, and (4) being more intuitive and graphical in our descriptions. The final decision was on actually going beyond normal surveys that only revise papers and do not offer too much added value. We intentionally tried to offer some contributions to make this a proactive and useful survey, in particular:

- We offer a chronological survey of articles and results in PGAs so the reader grasps how the trends are evolving at world level.

- We make an initial discussion on PGA models to make the survey pedagogical and self-contained (to an extent). This also helps to make better proposals for those interested in building models and later implement them in parallel platforms.
- We summarize the most cited papers in the last six years (fresh vision) and point out the most important venues (both journals and conferences) so researchers know where to find new papers in the future and where to head if interested in publishing their results on PGAs.
- We give a vision on the most important keywords in the domain to offer a comparative idea of the importance of topics at this moment.
- We analyze authors and how the domain is attracting attention of both new and experienced researchers.
- We analyze the higher impact papers (in the number of cites) in a semantic-consistent way, as we also delve into their implementation, platforms, programming languages, and comments on their contents relevant for other practitioners.
- We analyze PGAs from the point of view of the applications where they are used, thus showing where they are successful (by looking at what we include) and where are some opportunities for new applications (by thinking in what it is not found in recent references).
- We dare to list several future research lines where appealing challenges can be found, to help in defining new Ph.D. theses and master works involving PGAs.

This survey is structured in several sections. The next one will be devoted to a methodological description of PGAs. Then, Section 3 will develop important and recent venues for finding/publishing results with PGAs. Later, in Section 4, we address the details found in recent articles on PGAs. Finally, in Section 5, we describe our vision of future challenges and end the article with several conclusions.

2 PARALLEL GENETIC ALGORITHM

This section gives a brief overview of GA and PGA and shows a new and comprehensive taxonomy of PGA. Then, we introduce implementations of PGA, which are usually employed in conventional PGA research.

2.1 Overview

A genetic algorithm (GA) is a powerful metaheuristic inspired by Darwin's theory of natural evolution. Algorithm 1 shows a pseudo-code of a standard GA. A GA is a population-based algorithm in which many tentative solutions are managed. First, an initial population P_0 is constructed based on an initialization method (`initialization`). An initial population is (often) generated in a random uniform manner, while some heuristics such as Latin hypercube sampling method [69] or problem-specific approaches are employed to give better starting points for the search. Tentative solutions in the population are evaluated depending on the fitness function (`evaluation`), which assesses how much a tentative solution is fitted to a target problem. After that, the main procedure of the GA is executed. In the main procedure, new tentative solutions P'_t are generated through variation operators (`variation`) such as crossover and mutation, and their fitness is evaluated. Then, a next population P_{t+1} is constructed from the current population P_t and the newly generated tentative solutions P'_t (`replacement`). These procedures are repeated until a pre-defined termination criterion is satisfied, such as the quality of the desired solution, the maximum number of evaluations, or the computing time.

A GA is well suited to apply parallelization because of its population-based approach, where all solution candidates can be dealt with in parallel. When extending a sequential GA with parallel techniques, we have to consider several characteristics of the built PGA. A new and

comprehensive taxonomy of PGA is shown in Figure 1, because these considerations and characteristics are quite a few and need some structure. This taxonomy consists of six main and indispensable components of PGA: Implementation, Hardware, Software, API, Applications, and Problem domain, which are indicated at the first layer. Each component is represented with different colors: Implementation in pink, Hardware in blue, Software in green, API in orange, Applications in purple, and Problem domain in cyan. Deeper layers (sub-taxonomies) of each component present further details. Such classification is also presented in previous PGA surveys. In the survey of Alba et al. in 2002 [10], authors focus on PGA models, hardware architectures, and APIs used to implement PGAs. The survey of Alba et al. in 2013 [7] focuses on PGA models, their applications, hardware architectures, and software including APIs, although it discusses not only PGA, but also parallel metaheuristics, including trajectory-based approach and population-based ones. The survey of Talbi in 2015 [106], which mainly focuses on parallel evolutionary combinational optimization, divides parallel designs into three levels: algorithmic-level, iteration-level, and solution-level and discusses on hardware architecture. However, in his survey, software, applications, and problem domains are not taken into account. Our proposed taxonomy, in contrast to them, comprehensively incorporates these elements necessary for the realization of PGA and represents them all together in a single figure.

Implementation indicates how to implement a GA when going for any parallel methodology. This topic will be discussed soon in the next subsection. Hardware is an important factor of PGA, because it is the source of many important time gains and influences important operations such as population management or communication model. The most popular hardware actually employed in PGA literature is discussed in Section 4.2. Software and API are also important factors of PGA. In this taxonomy, Software consists of Library and Program Language. Several libraries for GA are presented by researchers, while the choice of program language highly relates to not only implementations of PGAs but also to their computing speed. API supports communication between CPU and/or threads and management of task assignment—all this essential for a PGA. Actually, some researchers do not use such libraries and APIs to implement their algorithms, but they may help researchers to quickly put their algorithm into practice. Software and API aspects are described in Section 4.3. Application focuses on the final application and deals with features typical of this real-world application, while Problem Domain classifies PGAs according to the fundamental aspects of the solved problem. Problem Domain is distinguished from the viewpoint of the number of objectives, its fitness dynamics, the number of constraints, solution representation, and where an instance is coming from. Applications and problem domain are discussed in Sections 4.4 and 4.5.

ALGORITHM 1: Pseudo-code of a genetic algorithm

```
t ← 0;
initialization(P_0);
evaluation(P_0);
while Termination criterion do
    P'_t ← variation(P_t);                          // Change existing tentative solutions
    evaluate(P'_t);      // Compute the relative quality of every solution in the population
    P_{t+1} ← replacement(P_t, P'_t);
    // Merge the auxiliary and previous populations and select a new better one
    t ← t + 1;
end
```

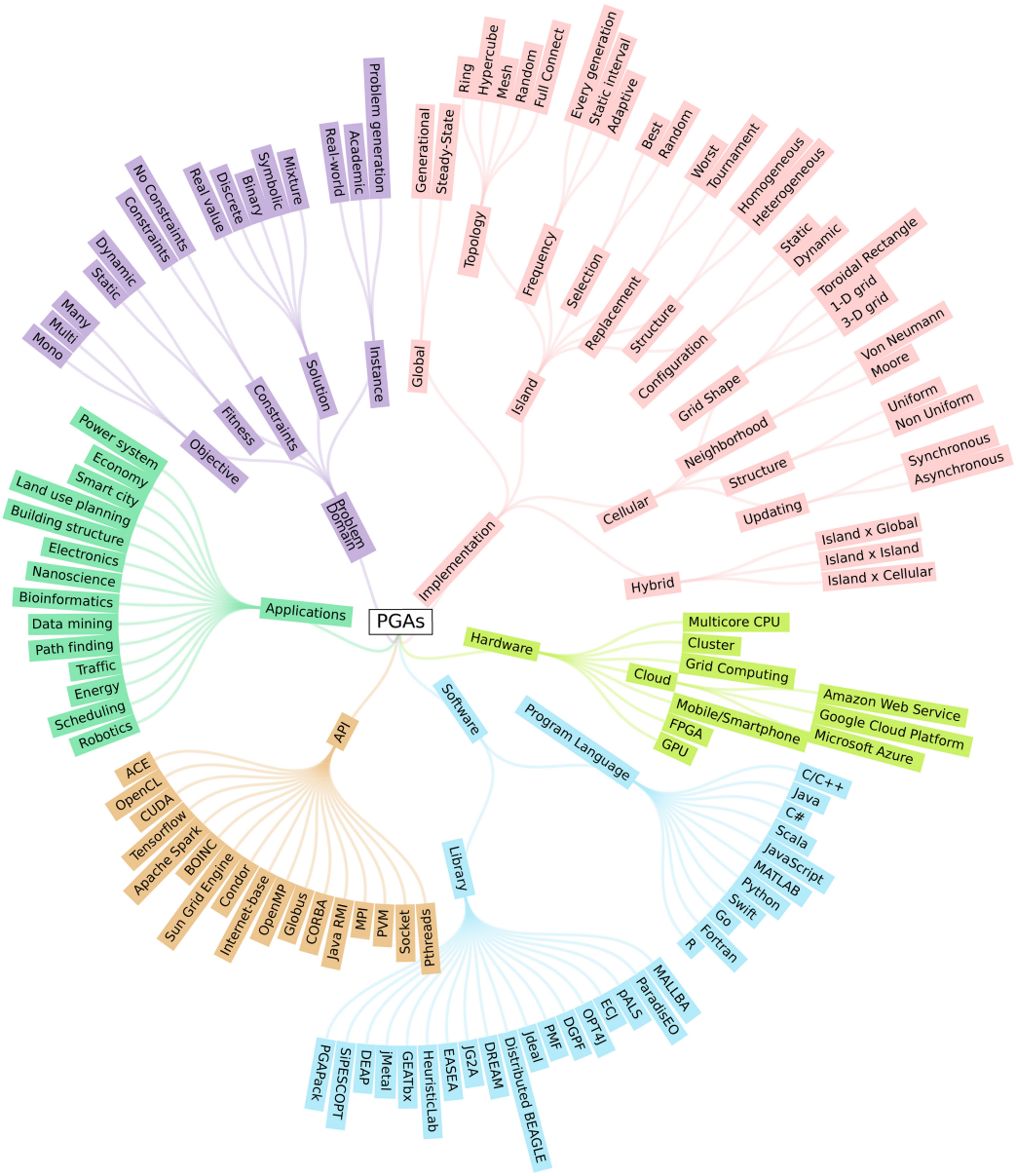


Fig. 1. A new comprehensive taxonomy of parallel genetic algorithms.

Regarding the implementation of a PGA, there exists mainly four kinds of PGA models: global parallelization, island model, cellular model, and hybrid models. The following subsections explain these details.

2.2 Global Parallelization

The global parallelization is the most simple implementation of a PGA. Figure 2(a) illustrates the global model. In the figure, gray circles indicate solutions. In the global model, tentative solutions

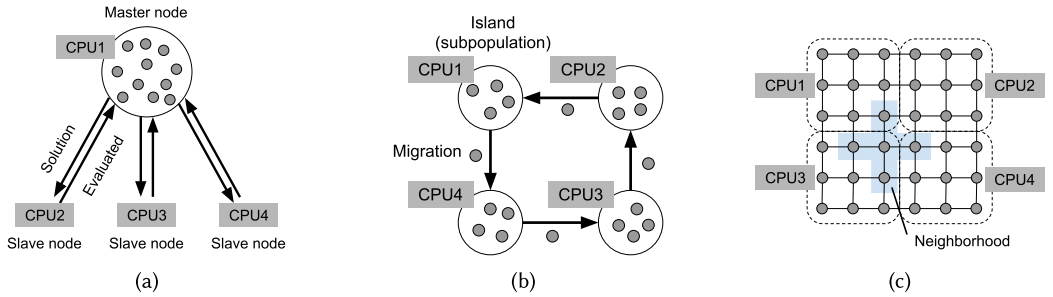


Fig. 2. Three usual PGA models: (a) global parallelization, (b) parallel island model, (c) parallel cellular model.

are managed a centralized population, and each solution is evaluated in parallel. After evaluating solutions, new tentative solutions are generated from the centralized population. The global model is generally classified into two further approaches: generational and steady-state. The generational approach generates a new population after all slave nodes completed their tasks. However, the steady-state approach generates a new solution immediately after one slave node completes its task and returns it as a result. Unlike the generational approach, which replaces one complete generation by another, the steady-state approach does not wait for all solution evaluations, and it generates and replaces a solution one-by-one. In the steady-state mode of operation, the GA merges new and old solutions continuously, yielding a usually faster convergence of the population with respect to generational models. Though this happens in global parallelization, it is also the main existing distinction for normal panmictic GAs where parallelism is not present.

The well-known master-slave parallel architecture is generally employed to implement the global model, where the master computing node manages the population and sends evaluation tasks to slave computing nodes. The slave computing nodes execute tasks given by the master node and return the expected results, e.g., the fitness of a received solution or the resulting individual, after altering it in some way (mutation is done in parallel, for example). Tasks handled with the slave nodes are usually the evaluation of solutions, because it is the most computationally expensive part in the GA procedure. In addition to the evaluation, variation operators are often executed on the slave nodes. Since the global model does not change the numerical behavior of a sequential (panmictic) GA, the same search ability can be expected between a sequential and global parallel GAs (of course faster in the case of the parallel GA).

2.3 Parallel Island Model

The parallel island (or multi-population) model is a PGA that manages several sub-populations in separate islands and executes the GA procedure in each island in parallel over a different set of solutions. Figure 2(b) shows a picture of the island model. Unlike the global parallelization, which manages a centralized population, the island model manages a set of decentralized populations. The decentralized populations are evolved separately and information of solutions in each island are exchanged at certain moments of evolution. This exchange of solutions is called a migration. The migration moves or exchanges (usually a copy of) solutions of each island to other ones depending on the topology of islands. Several topologies have been proposed; for example, ring, hypercube, mesh, random, and fully connect. The topology affects how migrated solutions are propagated to other islands and thus global convergence to the optimum. Regarding the migration, migration frequency and selection/replacement strategies must be configured. The migration

frequency determines the interval of the migration, while the selection/replacement strategies determine which solutions of each island are selected as the migrants or replaced with the migrants from other islands. Needless to say, all islands can follow a similar strategy for search (e.g., population size, the probability of genetic operators) and static configurations. However, to increase the diversity of solutions or to promote the convergence speed, heterogeneous policies and/or dynamic configurations can be employed. In the homogeneous structure, all islands have the same strategy for search, i.e., run the same algorithm in all islands, while in the heterogeneous structure, each island has different strategies—for example, different population size and/or different probability of genetic operators [51]—and in some heterogeneous structures, each island performs different EAs [27]. According to this, the optimization behavior of each island differs from each other, which contributes to increasing the diversity of solutions. Dynamics of configuration is another aspect of the island model. In the static configuration, a configuration of search is constant during the optimization process, while in the dynamic configuration, a configuration may change during the optimization process depending on the pre-defined manner or depends on the quality of solutions in each island.

To implement the parallel island model, each island is generally assigned to one computing element. For example, as shown in Figure 2(b), four CPUs are used to execute a parallel island model having four islands, each one managed by separate CPU where a GA on different population is running. At every migration step, a form of solution information is exchanged (migrated) to other CPU following some sort of communication strategy, e.g., shared memory or message passing, according to migration topology.

2.4 Parallel Cellular Model

The parallel cellular (or fine-grain) model arranges solutions on a grid structure and the variation procedure, and the replacements are executed to solutions within a defined neighborhood. Figure 2(c) illustrates the parallel cellular model. In the parallel cellular model, all solutions are arranged on a grid, and variation operators (e.g., crossover and mutation) and replacement are executed within certain neighborhoods of a grid. For example, in Figure 2(c), each solution is mated with solutions of the top, bottom, left, and right neighborhoods on a grid, and a generated offspring is replaced within its birth neighborhood. A toroidal rectangle is mostly employed as a grid topology, while other topologies can be considered, such as 1-D or 3-D grid shape. Several neighborhood strategies can be employed; for example, Von Neumann shown in Figure 2(c) is usually used, while a Moore neighborhood, in which solutions at top left, top, right, bottom left, and bottom right are neighbors in addition to the Von Neumann neighborhood, is also used. As it happens in the island model, a non-homogeneous structure can be employed in the cellular model; in particular, each solution on the grid can be managed according to different parameter settings and/or variation operators. Additionally, an updating strategy can be configured in the cellular model. There are synchronous and asynchronous updates of cells [49]. Synchronous updating updates all solutions simultaneously, while asynchronous updates cells at a time in some order. And in asynchronous updating, several updating strategies for the positions to be updated exist: fixed line sweep, fixed random sweep, new random sweep, and uniform choice [94]. In fixed line sweep, cells are updated sequentially from left to right and line after line starting from upper-left corner cell. In the fixed random sweep, the next cell to be updated is chosen with uniform probability, and the same permutation is then used for all update cycles while this new random sweep generates new random permutation for each update. Uniform choice always randomly selects cells to be updated with uniform probability.

Several implementations can be considered for the parallel cellular model. Figure 2(c) shows an example of implementations, where four CPUs are employed and each CPU manages a grid

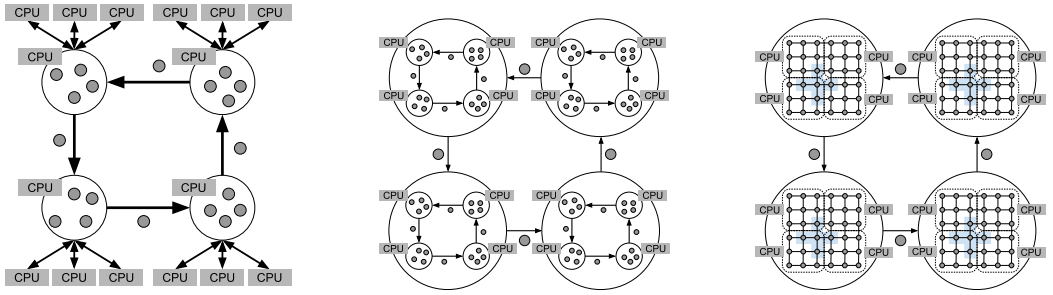


Fig. 3. Three examples of hybrid PGAs: (a) Island × Global, (b) Island × Island, (c) Island × Cellular.

including nine solutions. Four CPUs simultaneously execute selection, variation operators, and replacement following a defined neighborhood strategy. In this implementation, information of solutions on the edge of each grid has to be exchanged to each other through a physical communication network, because a part of neighborhoods is in a grid of other CPU. Other different and interesting implementations can be done when using a graphics processing unit (GPU); there, each solution is assigned to one core unit or thread in the GPU, and each processing unit executes selection, variation operators, and replacement for an assigned solution and its neighborhoods. Many issues happen here because of the synchronicity of GPUs and their need of making exactly the same operations if one wants to have a maximum speedup, so they say in which a cellular GA runs on a GPU is tricky [116].

2.5 Parallel Hybrid Model

The parallel hybrid approach combines two or more of the PGAs described before. We can see examples of hybrid models in Figure 3. Figure 3(a) shows a hybrid model of the island model and the global parallelization, in which the island model is used as a first level and each island executes the GA tasks of solution evaluations and/or variation operators in parallel. Figure 3(b) shows a hybrid model of the two island models, in which the island model is also used in a first level, and each island also executes the island model PGA. Figure 3(c) shows a hybrid model of the island model and the cellular model, in which the island model is also employed as a base model, while each island executes the cellular parallelization. Note that these are just examples and other hybrid approaches of combinations of PGA are possible.

3 ANALYSIS OF RECENT PGA STUDIES

In this section, we show an analysis and offer statistics on a set of recent PGA studies. We followed a systematic literature review protocol as the one described in References [58, 59]. First, we decided to define the research questions that this survey is meant to answer:

RQ1: How is the PGA field evolving? How many PGA papers were published and which journals/conferences PGA researchers are focusing on?
RQ2: What research topics are hot in recent PGA works?
RQ3: Which individuals and countries are most active in recent PGA works and how are they related?
RQ4: What domains are PGAs applied to and how is this done?

To answer these research questions, this survey considers papers related to PGAs from IEEE Xplore,[1] ACM Digital Library,[2] Springer Link,[3] and Scopus.[4] These scientific web services contain most important journal papers and conference proceedings, not only in the EC domain, but also in the computer science and engineering domains. They were found after a careful search by using several keywords related to PGAs. Before conducting our survey, we first discussed keywords related to PGA algorithms and ones related to parallelism. In particular, we chose the words *parallel, island model, island based, concurrent, fine-grain, coarse-grain, master-slave, subpop, sub-pop, distributed* as keywords related to the algorithm itself, while for parallelism, we chose the words *cluster, GPU, graphics processing unit, FPGA, programmable gate array, multi-core, multicore*. We then determined the search query that combined the above keywords with the word "genetic algorithm" or "ga." To avoid finding irrelevant papers, we modified the Boolean operators while confirming the titles in the search result. After this modification, we finally decided the search query for each repository. The detailed search query used for each web service is shown in the supplemental material. The search year was specified from January 1st, 2013, to December 23rd, 2018. First, we got 469 papers from IEEE Xplore, 294 papers from ACM Digital Library, 124 papers from Springer Link, and 994 papers from Scopus. In total, we got 1,881 papers. Then, duplicated papers were automatically removed. Finally, we have analyzed a total of 1,280 papers, including 636 journal papers and 644 conference papers. We read the main contents relevant from this survey in the resulting set of papers after our initial filtering of literature.

Section 3.1 first analyzes the statistics of the number of publications for each year and identifies the journals and conferences publishing more PGA articles in the time frame studied. Then, Section 3.2 analyzes the keywords used by these PGA papers (a semantic study of the state-of-the-art), while Section 3.3 discusses the authorship of the collected papers (a strategic analysis of the state-of-the-art). Finally, Section 3.4 summarizes all the findings of this first part of our survey, focusing on how PGA articles (and venues) developed themselves in past years.

3.1 Publications

In this section, we discuss statistics of publications of PGA works. First, we show the change in the number of publications within the last six years. Then, we show the detail of publications from the viewpoints of journal articles and conference papers and discuss what journals/conferences PGA research are focusing on. This analysis helps new master and Ph.D. researchers to find which journals/conferences should be considered and focused on to learn recent trend of PGAs and are suited to submit their new PGA papers.

Figure 4 shows the number of publications related to PGAs that appeared in both journals and conferences in the last six years. The horizontal axis indicates the year, while the vertical axis indicates the number of publications for each year. The blue bar with horizontal striped lines indicates the number of journal papers, while the orange bar with diagonal striped lines indicates the number of conference papers and chapters in books. From this histogram, it can be found that almost 200 papers related to PGAs are published every year, for a total of 1,280 papers in the last six years. We can notice that the total number of papers has slightly decreased in the last two years, what we interpret as a shift to interest to the applications of PGAs instead of going to the fundamentals of these algorithms. This is because many researchers recently pay attention to applications of PGA, but not to fundamental or methodology of it, and their titles, keywords, and

[1] https://ieeexplore.ieee.org/Xplore/home.jsp.
[2] https://dl.acm.org/.
[3] https://link.springer.com/.
[4] https://www.scopus.com/wsearch/form.uri?display=basic.

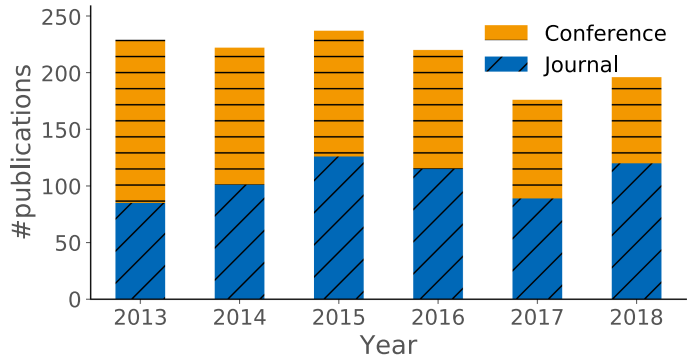


Fig. 4. The number of publications related to PGAs from 2013 to 2018.

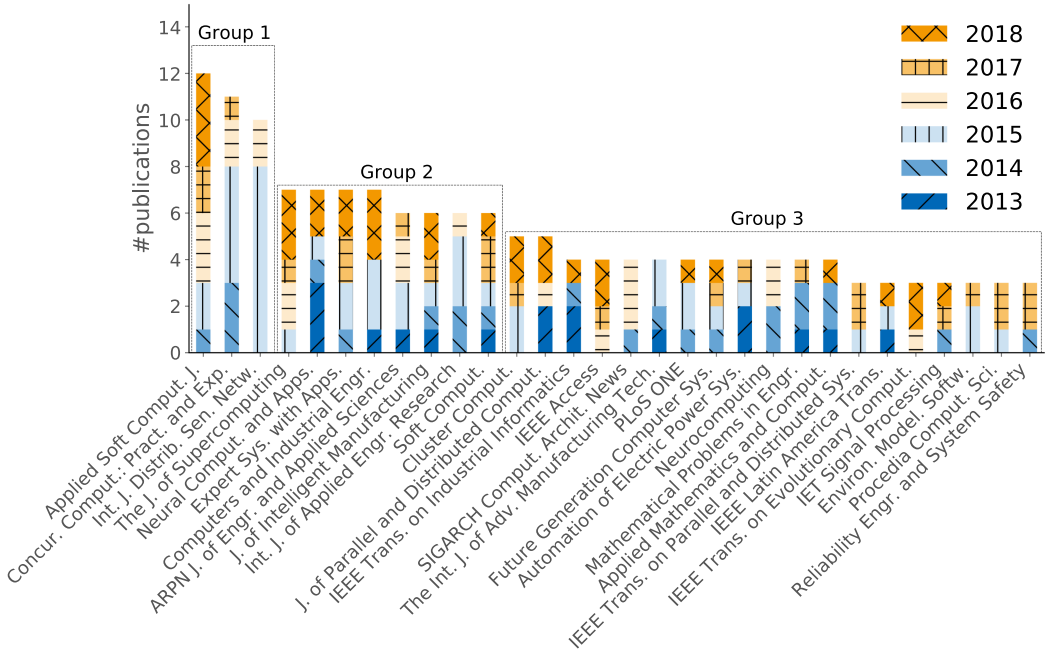


Fig. 5. Journals that have the largest number of PGA papers in 2013–2018 (only 30 journals included).

abstracts do not contain words related to PGA explicitly. Anyway, it is not even a trend (so small difference), since the changes along years could be also interpreted as the normal fluctuation in the publishing of a large international community of researchers.

Figure 5 shows the journal papers that have the maximum number of PGA papers, and the horizontal axis indicates the name of the 30 top journals, while the vertical axis indicates the number of publications. From this figure, we can see there are mainly three groups of journals, focusing on the number of publications and the scope of journals. The first group consists of the top three number of journals. One of them, *Applied Soft Computing Journal*, is related to soft computing including GA, while two of them, *Concurrency and Computation: Practice and Experience* and *International Journal of Distributed Sensor Networks*, are related to the parallel system. This indicates that many of the PGA papers published were submitted focusing on the topics of parallel systems

Table 1. Top 10 Journals

Rank	Journal	Total	2013	2014	2015	2016	2017	2018
1	Applied Soft Computing Journal	12	0	1	2	3	2	4
2	Concurrency and Computation: Practice and Experience	11	0	3	5	2	1	0
3	International Journal of Distributed Sensor Networks	10	0	0	8	2	0	0
4	The Journal of Supercomputing	7	0	0	1	2	1	3
4	Neural Computing and Applications	7	3	1	1	0	0	2
4	Expert Systems with Applications	7	0	1	2	0	2	2
4	Computers and Industrial Engineering	7	1	0	3	0	0	3
8	ARPN Journal of Engineering and Applied Sciences	6	1	0	2	2	1	0
8	Journal of Intelligent Manufacturing	6	1	1	1	0	1	2
8	International Journal of Applied Engineering Research	6	0	2	3	1	0	0
8	Soft Computing	6	1	1	1	0	2	1

and soft computing algorithms. The second group, from 4th to 11th ranks, contains journals related to applications or manufacturing. Actually, six journals out of these eight ones are related to applications or manufacturing, in particular, *Neural Computing and Applications* (4th rank, seven publications), *International Journal of Applied Engineering Research* (4th rank, seven publications), *Expert Systems with Applications* (4th rank, seven publications), *Computers and Industrial Engineering* (4th rank, seven journals), *ARPN Journal of Engineering and Applied Sciences* (8th rank, six publications), *Journal of Intelligent Manufacturing* (8th rank, six publications), and *International Journal of Applied Engineering Research* (8th rank, six publications). This indicates that many applications of PGAs are published in journals related to applications and manufacturing, and they are submitted to journals that well match to the topic of specific but of a different (assorted) kind applications. The third group, from 12th to 30th ranks, has the similar trend: Many journals are related to applications, such as *IEEE Transactions on Industrial Informatics* (14th, four publications), *the International Journal of Advanced Manufacturing Technology* (14th rank, four publications), *Automation of Electric Power Systems* (14th rank, four publications), *Mathematical Problems in Engineering* (14th rank, four publications), *Applied Mathematics and Computation* (14th rank, four publications), and *IET Signal Processing* (24th rank, three publications). Since journals regarding applications, industrial, and manufacturing are ranked in the top 10 list, it seems we can suggest that there is a recent shift in attention to applications of PGAs rather than their fundamentals. In particular, *IEEE Transactions on Evolutionary Computation*, *IEEE Transactions on Parallel and Distributed Systems*, and *the Journal of Parallel and Distributed Computing*, well-known journals in the EC and the parallelism domains, have few numbers of publications related to PGAs; indeed, 3, 3, 5 papers, respectively.

Table 1 shows the details for the top 10 journals (since 8th rank is a tie, 11 journals are shown in Table 1). The "Journal" column indicates the name of a journal, while the "Total" one indicates the total number of publications for each journal; the rest of the columns indicate a breakdown for each year. From this table, we can first observe that *International Journal of Distributed Sensor Networks* publishes PGA papers only in 2015 and 2016, while no paper is published in 2013, 2014, 2017, or 2018. However, the other journals constantly publish one or more PGA papers every year. This indicates that, although the number of journal publications of PGA works is not high, PGAs are constantly paid attention in the recent six years not only in the soft computing or the parallel system domain, but also in the domain of applications. We guess that *Applied Soft Computing*

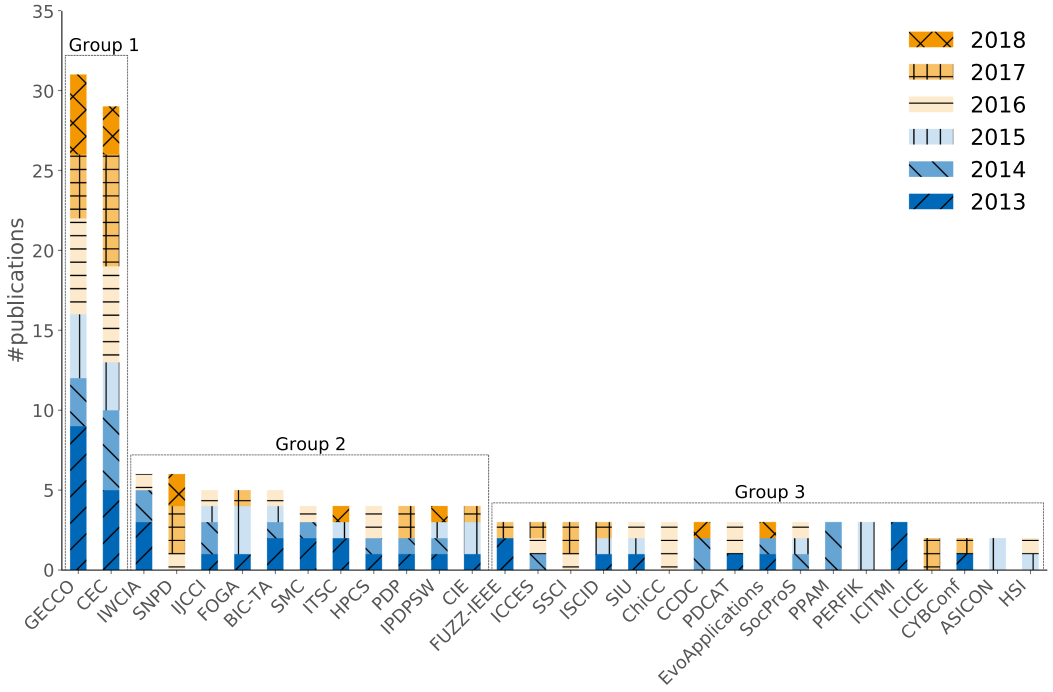


Fig. 6. The number of conference papers and chapters in books from 2013 to 2018 (top 30).

Journal is a preferred venue for PGAs because (1) its focus is wider with respect to journals on parallelism, so more researchers publish and cite papers there, (2) their total time for publication has been shortened along these last years, and (3) it allows both theory and applied papers.

Figure 6 shows the papers in the conference proceedings and chapters in books that have the top 30 number of publications. Axes and colors have the same meaning as in Figure 5. From Figure 6, we can see also three groups of conferences, focusing on the number of publications and the scope of conferences. The first group consists of two conferences, *ACM Genetic and Evolutionary Computation Conference (ACM GECCO)* and *IEEE Congress on Evolutionary Computation (IEEE CEC)*, which definitely have the larger number of publications in the PGA domain. Actually, these two conferences have a high impact not only in the PGA domain, but also in the whole field of evolutionary computations, so it is still important for PGA researchers to gather information from and present new results at these two major conferences. The second group consists of 11 conferences: *IWCIA*, *SNPD*, *IJCCI*, *FOGA*, *BIC-TA*, *SMC*, *ITSC*, *HPCS*, *PDP*, *IPDPSW*, and *CIE*. Notable here is that 6 conferences out of 11—*IWCIA*, *SNPD*, *BIC-TA*, *SMC*, *ITSC*, and *CIE*—contain the terms "applications," "engineering," or "system" in the name of conferences and focus on an area of applications of evolutionary computation and computational intelligence, including PGA. This indicates that the PGA works applied to real-world problems have been published in rest of conferences that focus on the applications of evolutionary computation and computational intelligence. While 3 conferences out of 11—*HPCS*, *PDP*, and *IPDPSW*—related to the parallel system. This indicates that some PGA research is mainly interested in the parallel system as a first target for the studies. The rest of this group, *FOGA* and *IJCCI*, are related to GA and computational intelligence. A finding here is that PGA works have been published in three domains—evolutionary computation, applications, and the parallel system—in these two groups, uniformly. In contrast to the previous two groups, the third group consists of conferences that stress the application of

Table 2. Top 10 Conferences

Rank	Conference/Chapter in books	Total	2013	2014	2015	2016	2017	2018
1	GECCO	31	9	3	4	6	4	5
2	CEC	29	5	5	3	6	7	3
3	IWCIA	6	3	2	0	1	0	0
3	SNPD	6	0	0	0	1	3	2
5	IJCCI	5	1	2	1	1	0	0
5	FOGA	5	1	0	3	0	1	0
5	BIC-TA	5	2	1	1	1	0	0
8	SMC	4	2	1	0	1	0	0
8	ITSC	4	2	0	1	0	0	1
8	HPCS	4	1	1	0	2	0	0
8	PDP	4	1	1	0	0	2	0
8	IPDPSW	4	1	1	1	0	0	1
8	CIE	4	1	0	2	0	1	0

intelligent techniques. In particular, five conferences (*ICCES*, *SIU*, *EvoApplications*, *ICITMI*, and *ICICE*) mainly focus on applications, but not always on the evolutionary computation.

Table 2 shows the details of top 10 conferences and chapters in books (since we have a tie in 8th position of the ranking, 13 conferences or chapters in books are shown in Table 2). Each column has the same meaning as Table 1. From this analysis, it can be confirmed that two major conferences in the EC domain—*ACM GECCO* and *IEEE CEC*—have high importance for the PGA domain (as any well-informed researcher could expect). Actually, these two conferences (*ACM GECCO* and *IEEE CEC*) account for 4.8% and 4.5% of publications in all conference papers related to PGA, respectively. In both conferences, PGA topics are constantly presented every year.

In this section, we have then answered RQ1 by analyzing the evolution of the number of publications within the last six yeas and by finding out the preferred venues to publish on PGAs.

3.2 Keywords

This section discusses the keywords used in PGA papers as a way of answering RQ2 (hot topics in PGAs). We have built a word cloud of keywords in PGA papers and discuss hot topics of recent PGA works by analyzing what keywords are frequently used.

Figure 7 shows the word cloud we have generated by using keywords in the collected papers. The plural and case forms of the terms in the cloud are filtered, so "genetic algorithm," "Genetic Algorithm," and "genetic algorithms" are all treated as "genetic algorithm." Larger words in the word cloud are the ones used more frequently in the papers analyzed.

From this word cloud, although keywords related to the algorithm such as "genetic algorithm," "parallel genetic algorithm," "parallel computing," and "optimization" are understandably used more frequently, it is notable that the keyword "multi-objective" has been attracting researchers attention as a problem domain of PGA. In the last decade, since just as research in multi-objective GA (MOGA) increased in the EC domain, their parallelization also became an important topic. As a keyword related to hardware, terms related to GPU, such as "graphics processing unit" and "cuda," are also frequently found. GPU recently became a very important device, not only in the EC research, but also for other domains like deep learning. The reason is the big computation power that can be drawn from them to compute many tasks simultaneously. Regarding the problem domain, "scheduling" is frequently used as a keyword in PGA papers. This is because many applications of

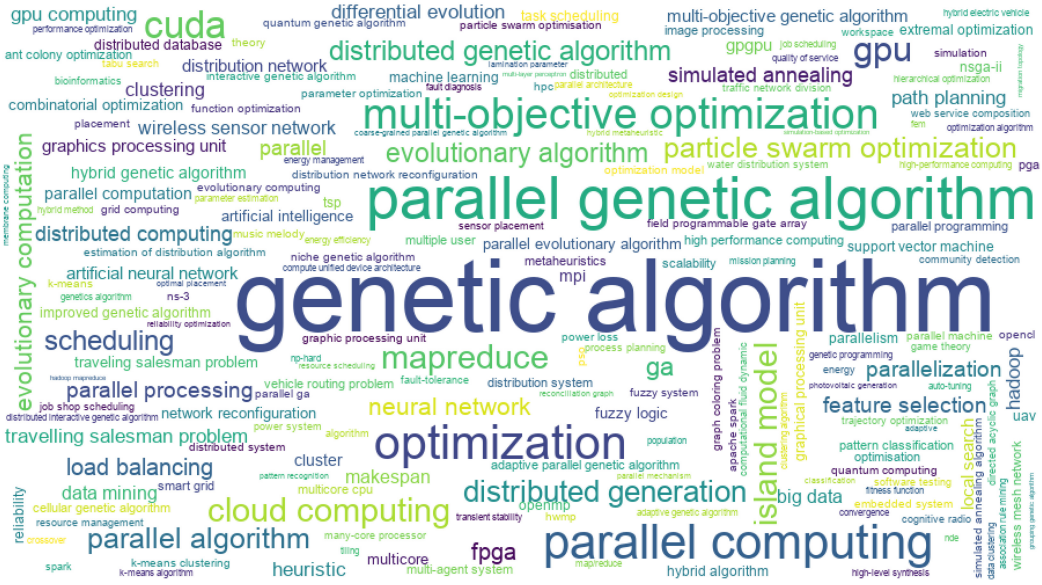


Fig. 7. Word cloud of keywords in PGA papers.

scheduling can be found as applications (for example, transportation, manufacturing, or task assignment) and even as internal ways of implementing PGAs (e.g., scheduling the parallel processes running the PGA).

3.3 Authorship

In this section, we analyze the authorship of PGA research as a way of answering RQ3 on the social aspect of research in PGAs. First, we discuss the change in the number of authors in PGA research in the last six years. Additionally, we show the percentage of newcomers every year. Then, the number of works per each author and the number of co-authors in each PGA paper are shown. And finally, we show countries where authors of PGA research are from and their relationship from the point of view of co-authorship. The final aim is to give a strategic vision of how the PGA field develops these days.

Figure 8 shows the cumulative number of authors who published PGA papers from 2013 to 2018. The horizontal axis represents the year, while the vertical axis shows the cumulative number of authors. From this figure, the cumulative number of authors constantly increases every year from 2013 to 2018, and in total, 3,961 authors wrote PGA papers from 2013 to 2018.

Figure 9 shows the percentage of new authors who first published PGA paper(s) as (co-)author per year from 2013 to 2018. The horizontal axis represents the year, while the vertical axis shows the percentage of new authors who first published PGA paper(s) per each year. Here, "new author" accounts for authors who first published PGA paper(s) in the years during 2013 to 2018, while the number of "active author" accounts for authors who published PGA paper(s) whether the authors are new or not. Since all authors are new and active in 2013 in this survey, the percentage in 2013 is 100%. In 2014–2018, the percentages of the new-to-active author are relatively high, in particular, 81.9%–92.5%. This indicates that most of the authors constantly join the community but only publish a few times.

Table 3 shows the number of authors that published each number of works. The "# Authors" row indicates the number of authors who publish papers of the number of the corresponding columns,

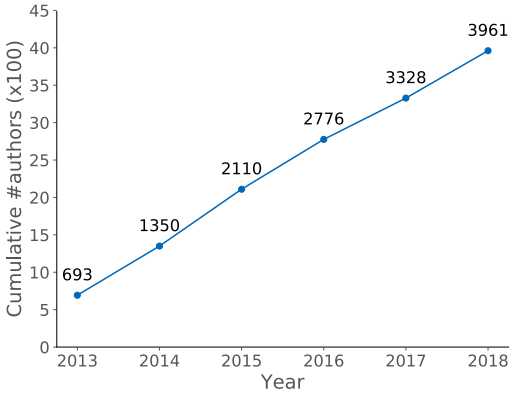


Fig. 8. Cumulative number of authors.

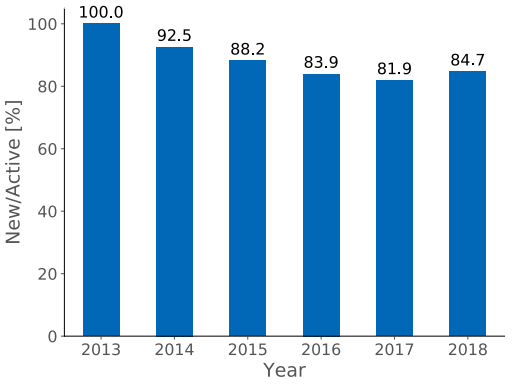


Fig. 9. Percentage of new authors per year.

Table 3. The Number of Authors Who Published Each Number of Works and Its Percentage

# Works	1	2	3	4	5	6	7	8	9
# Authors	3,071	344	72	13	8	6	1	2	1
%	87.29%	9.78%	2.05%	0.37%	0.23%	0.17%	0.03%	0.06%	0.03%

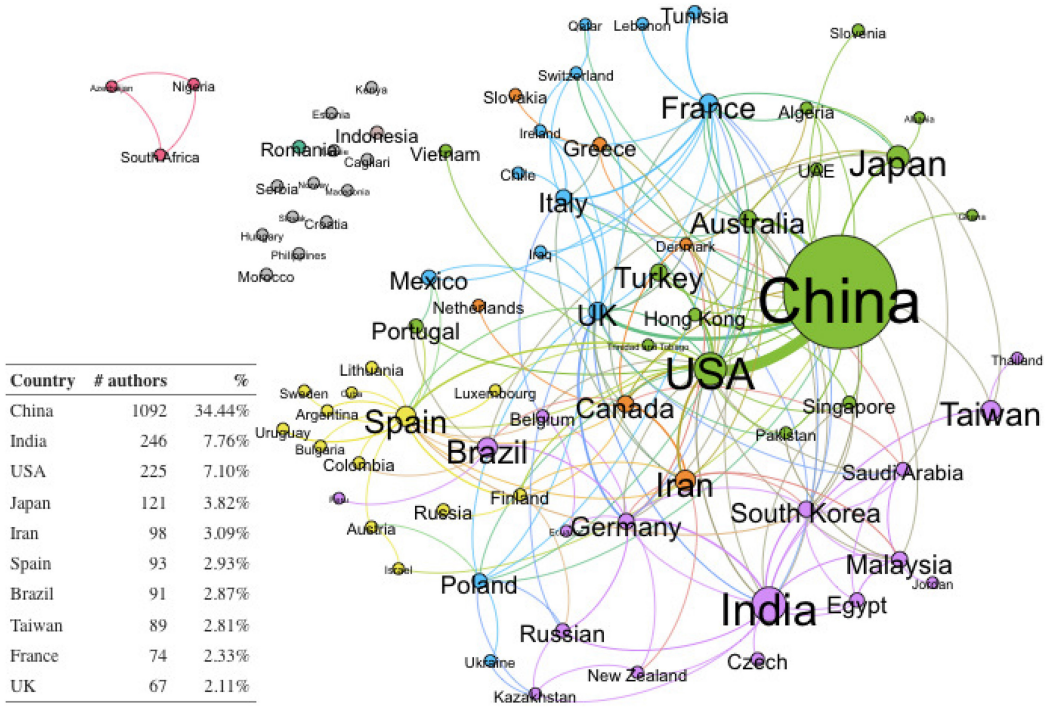
Table 4. The Number of Co-authors for Each Paper and Its Percentage

# Co-authors	1	2	3	4	5+
# Works	85	340	351	255	230
%	6.74%	26.96%	27.84%	20.22%	18.24%

while the "%" row indicates its percentage in the total number of authors. 87.29% authors published only one PGA paper, while 9.78% published two papers from 2013 to 2018; in total, 97.07% of authors published two papers or less. This analysis supports the result of Figure 9 in which most of the authors are newcomers to the PGA domain. This is one reason for the sometimes ill utilization of previous literature of these newcomers, who miss important papers and concepts when writing on PGA, and the effect we have detected frequently in this community.

Table 4 shows the statistics of authorship for each paper. The "# Co-authors" row indicates the number of authors per paper, while the "# Works" and "%" rows indicate the number of works with authors of the number of corresponding column and its percentage in the total number of works. In this table, we can see that most works are done by two or three authors, while only 18.24% of works are done by five or more authors.

Figure 10 shows the number of authors from each country and region and the relationship between countries from the viewpoint of co-authorship of the PGA papers; that is, international collaborations in PGAs. In the left bottom table, the "Country" column indicates countries, while the "# authors" and "%" columns indicate the number of authors from a given country corresponding to each row and its percentage. China is the country where the higher number of authors work in the PGA domain during the last six years, and its percentage is about 34% of all world authors. After China, authors from India and USA account for about 7.5% of all authors. Subsequently to



Country	# authors	%
China	1092	34.44%
India	246	7.76%
USA	225	7.10%
Japan	121	3.82%
Iran	98	3.09%
Spain	93	2.93%
Brazil	91	2.87%
Taiwan	89	2.81%
France	74	2.33%
UK	67	2.11%

Fig. 10. The number of authors from each country and region (top 10) and country relationship of co-authorship.

them, authors from Japan, Iran, Spain, Brazil, Taiwan, and the UK account for 2% –4% of all authors, and totally about 69% of authors are from these top 10 countries and regions.

In the relationship graph in Figure 10, the size of circles indicates the number of authors, while the width of edges indicates the number of co-authored papers between two countries. The color of circles indicates the difference in communities that is found by a community analysis. From this analysis, we find out that the most frequent co-authorship of two countries are China and the USA, and these countries have many co-authorships with several countries regardless of geographical zone. Other than this, several communities of co-authorship of countries can be found. For example, Spain has connections to many countries in Europe and South America, such as Brazil, Portugal, Argentina, Colombia, Luxembourg, Finland, and so on. Focusing on India, there are many co-authorships between Asian countries; for example, South Korea, Taiwan, Singapore, Malaysia, and so on. In Europe, UK, France, and Germany work with authors of many countries not only in Europe but also in Asia, North America, and South America. From these analyses, in the PGA domain, there are many co-authorships of researchers not only in certain countries but also in different geographical zones.

Now that we have concluded on researchers and countries, let us make a summary in the next section of all the findings done till now.

3.4 Summary

In this whole section, we showed an analysis and offered statistics on a set of recent PGA studies. We analyzed publications of PGAs from three points of view: the number of publications

throughout the year, the keyword used in papers, and authorship. This section is then summarized as follows:

- The number of publications related to PGAs decreases in the last six years. Additionally, many researchers pay attention to applications of PGAs, but not basics or fundamentals.
- From the analysis of keywords appeared in PGA papers, it is indicated that "multi-objective," "graphics processing unit," and "scheduling" are hot topics in this domain.
- About 4K researchers joined in the PGA domain and more than 80% of researchers are new ones in the last six years. Most works are collaborations between two or more authors. And many works are treated not only by authors in one country but also by a collaboration of several countries.

4 LITERATURE REVIEW

In this section, we will concentrate on analyzing the related recent literature to answer our final research question RQ4 (What domains are PGAs applied to, and how has this been done?). In particular, we mainly focus on papers that are the most cited 20 papers (which are cited more than 24 times). Table 5 shows a list of papers that are the most cited 20 papers.[5] In addition to them, we provide additional descriptions of more recent papers related to each topic even though they are cited less than 24 times. In this table, the "Ref." column indicates the reference of papers, while the "Cites" column indicates the citations of each paper. The "Impl." column indicates the employed implementation of PGA in each paper. The "HW" column indicates the hardware platforms used to implement PGAs, while the "# processors" column shows the maximum number of processors actually used in the experiment. The columns labeled "API," "Library," and "Prog. lang." show APIs, Libraries, and programming languages used to implement PGAs, respectively. The column "Prob. instance" finally indicates problem instances employed in papers, in which "Academic" represents academic standard benchmark problem instances, "Problem generation" represents problem instances randomly generated, and "Application" represents that PGA is applied to real-world applications in that paper. If such information is not clearly indicated or not available in the papers, the cell is filled with "-".

Section 4.1 discusses the implementation of PGAs employed and studied in the listed papers. Sections 4.2 and 4.3 describe the hardware and software used in the listed papers, respectively. Section 4.4 describes academic problem instances and problem generators, and finally, Section 4.5 describes real-world applications solved by PGAs.

4.1 Parallel Implementation

We start this section with a discussion on papers from the point of view of the PGA implementation that we described in Section 2. From the implementation aspect, the parallel island model and the global parallelization are the majority (17 papers out of 20), while 1 paper employs parallel cellular model (clearly more research in this domain is advisable). Three papers employ a parallel hybrid approach. These details are described in the following:

Parallel island model. Table 6 summarizes the characteristics of the parallel island models used in the papers in Table 5. The "Topology" column indicates the communication topology used in that paper, while the "Migration Frequency" column indicates how frequently solution migration is conducted. The "Selection" and "Replacement" columns indicate how to select migrated solution(s) from a source subpopulation (island) and how to select the replaced solution with migrated solution(s) in the target subpopulation, respectively. The "Configuration" column indicates

[5]The number of citations was checked in Scopus at December 23rd, 2018.

Table 5. The Most Cited 20 Papers (Cited More Than 24 Times)

Ref.	Cites	Impl.	HW	# processors	API	Library	Prog. lang.	Prob. instance
Roberge et al. [88]	265	Island	Multicore CPU	8	Parallel Computing Toolbox	-	MATLAB	Real-world
Devos et al. [38]	58	Global	Cluster	16	Distributed Computing Server	GEATbx	-	Real-world
Liu et al. [66]	49	Global	Cluster	30	MPI	-	Fortran	Real-world
Sahingoz [90]	41	Global	Multicore CPU	8 (with HT)	-	-	Java	Real-world
Porta et al. [80]	41	Global	Mulcicore CPU	8	java.util.concurrent	-	Java	Real-world
		Island	Cluster	8	MPJ			
		Hybrid	Hybrid	32	MPJ			
Shayeghi et al. [96]	38	Hybrid	-	(up to) 240	-		-	Real-world
Wen et al. [118]	37	Island	Cloud	300	MPI	-	-	Problem generation
Roberge et al. [89]	37	Hybrid	GPU	512	CUDA	-	-	Real-world
Kurdi [61]	33	Island	Multicore CPU	3	-	-	-	Academic
Yang et al. [124]	29	Global	Cluster	48	Node.js, HTCondor	SiPESC.OPT	JavaScript	Real-world
Allahviranloo et al. [14]	29	Island	Cluster	12	Parallel Computing Toolbox	-	MATLAB	Problem generation
Hong et al. [54]	28	Global	Cluster	(up to)130	-	-	Java	Problem generation
Pinel et al. [78]	28	Cellular	GPU	448	CUDA	-	-	Problem generation
Tosun et al. [110]	27	Island	Cluster	240	OpenMPI	-	C++	Academic
Ghosn et al. [48]	26	Island	Cluster	-	MPI	-	C	Academic
Soufan et al. [99]	26	Global	Cluster	64	MPI	PGAPack	MATLAB	Real-world
Ye and Huang [126]	25	Global	Cluster	32	MPI	-	-	Real-world
Zhang et al. [127]	25	Island	Cluster	100	MPI	-	-	Problem generation
Dorronsoro et al. [39]	24	Island	Multicore CPU	8	-	jMetal	Java	Problem generation
Cao and Ye [26]	24	Island	Multicore CPU	4	-	-	-	Real-world

"-" in the cell in this table indicates that this information is not available in the corresponding paper.

Table 6. Type of island model used in the most cited 20 papers

Ref.	Topology	Migration frequency	Selection	Replacement	Configuration	Structure
Roberge et al. [88]	Full connect	10 times	Random permutation of a global population		Static	Uniform
Porta et al. [80][a]	Full connect	{3, 6, 15} minutes	New population with best solutions from each island		Static	Nonuniform
Wen et al. [118]	Full connect	Every generation	New population with best solutions from each island		Static	Uniform
Kurdi [61]	Full connect	Adaptive[b]	$MigCou^{th}$ worst[c]	Worst	Dynamic	Nonuniform
Allahviranloo et al. [14]	Full connect	Every generation	Best	Worst	Static	Uniform
Tosun et al. [110]	Full connect	Every generation	Best	No replacement	Static	Uniform
Ghosn et al. [48]	Local/full connect	Every generation	Best	-	Static	Uniform
Zhang et al. [127]	Full connect	Adaptive[d]	Marge local solutions to global one		Static	Nonuniform
Dorronsoro et al. [39]	Full connect	Every generation	Best	No replacement	Static	Nonuniform
Cao and Ye [26]	Full connect	10 generations	-	-	Static	Uniform

[a] Cluster version.
[b] Migrate if there are no improvements in all of the islands for 10 successive generations.
[c] $MigCou$ is the count of migrations occurred.
[d] The number of fitness evaluations reaches to upper limit or better solutions are achieved.

whether the configuration of genetic parameters, e.g., mutation rate or migration frequency, are static or dynamically changed during the evolution process. The "Structure" column indicates that each subpopulation has a uniform (same) structure or nonuniform one, e.g., different mutation rates or different migration strategies in every subpopulation.

Most papers employ a fully connected topology where all subpopulations exchange information with each other and immigrants move to all other subpopulations. The work of Ghosn et al. [48] adopts a hybrid approach that allocates subpopulations to some groups, and a group of subpopulations locally shares their best solutions every G_N generations. However, every $L_N \gg G_N$ generation, the best solutions of all subpopulations are broadcasted to all others. This approach enables to reduce the communications overhead while ensuring a global broadcast of solutions.

Several migration frequencies are found in the existing literature, and some papers migrate solutions every generation, while others use the migration frequency based on the number of generations. The work of Porta et al. [80] employs wall-clock time as the migration frequency, independent of the number of generations. The work of Kurdi [61] decides the migration if there is no improvement in all of the islands.

For the selection strategy, four papers select the best solution to be migrated, while for the replacement strategy, two papers replace the worst solution with the incoming migrants. In the work of Roberge et al. [88], populations of all islands are merged and new populations are generated by randomly permutating and splitting the merged population. The work of Porta et al. [80] constructs a new population with the best solutions from each island and all islands start from a single global population. Similar to this, in the work of Wen et al. [118], local elite (best) results of each island are collected and synchronized into a global elite list, and the global elite list is broadcast to all islands. In the work of Kurdi [61], a new naturally inspired migration strategy is proposed in which $MigCou^{th}$ worst individual ($MigCou$ is the number of migration occurs in the optimization process) in each island are selected to be migrated to other islands.

Some papers use nonuniform configuration of islands, i.e., different parameters and/or policies are applied to each island. In the work of Kurdi [61], each island employs a different self-adaptation method as nonuniform configuration. The work of Dorronsoro et al. [39] proposes island-based cooperative coevolution multi-objective evolutionary algorithms (CCMOEAs). In this work, three kinds of CCMOEAs are compared; in particular, NSGA-II, SPEA2, and MOCell. In the proposed model, each subpopulation optimizes partial solutions, and the best partial solutions are migrated to other islands where they are utilized to evaluate partial solutions. The experimental study revealed that the parallel CCMOCell with four or eight subpopulations achieves better performance with faster computation time than other two CCMOEAs, though the solution quality of CCMOCell is slightly worse than the sequential MOCell. The work of Zhang et al. [127] deals with the set cover problem of wireless sensor networks, and their PGA uses a divide-and-conquer strategy. First, their PGA solves a small set of cover problem for equally divided target area on subpopulations where the selection, crossover, mutation, and fitness evaluation are executed within the subarea. Every f_s fitness evaluations, if each subpopulation achieves better solutions, local solutions are merged by using the Kuhn–Munkres (KM) algorithm to calculate the termination condition. These divide-and-conquer processes are repeated until the termination condition.

Global parallelization. Table 7 summarizes the feature of the global parallelization used in the papers in Table 5. The "Reproduction" column indicates whether the generational (synchronous) reproduction or the steady-state (asynchronous) reproduction is used. The "Slave's task" column indicates what operations the slave nodes execute in the evolution process.

All papers use the generational (synchronous) approach. This is because the generational approach can be directly designed based on conventional non-parallel GAs, and synchronous communication between master and slave nodes is easy to implement. As the characteristic of the

Table 7. Type of Global Parallelization Used in the Most Cited 20 Papers

Ref.	Reproduction	Slave's task
Devos et al. [38]	Generational	Evaluation
Liu et al. [66]	Generational	Evaluation
Sahingoz [90]	Generational	Selection, variation, and evaluation
Porta et al. [80][a]	Generational	Selection, variation, and evaluation
Yang et al. [124]	Generational	Evaluation
Hong et al. [54]	Generational	Evaluation
Soufan et al. [99]	Generational	Evaluation
Ye and Huang [126]	Generational	Evaluation

[a]Multicore CPU version.

Table 8. Hybrid Approaches Used in the Most Cited 20 Papers

Ref.	Hybrid	Detail
Porta et al. [80][a]	Island × Global	Islands are fully connected, and each island evolves by executing a new parallel level of global parallelization
Shayeghi et al. [96]	Global × Global	Pool-based steady-state global parallelization is employed, and a newly generated solution is evaluated in parallel
Roberge et al. [89]	Island × Global	Islands are connected with bidirectional ring topology, and each solution in island is evaluated in parallel

[a]Hybrid version.

global parallelization, the number of cores or processors used is more than that of the island model, which can be confirmed in Table 5. Since the number of executions of solution evaluations and/or genetic operations can be increased by increasing the number of processors in the global parallelization, it is suitable for parallelization on the system having many processors.

In the work of Liu et al. [66], global parallelization is used to evaluate solutions by performing a generalized probit-based stochastic user equilibrium traffic assignment, a framework for the route choice problem. An HPC system in the Civil and Environmental Engineering department at the National University of Singapore is used in this paper, and the maximum 30 processors are used to parallelize the solution evaluations.

The work of Sahingzo [90] solves the UAV path-finding problem with a PGA. In this work, paths of multiple UAVs are optimized simultaneously. Global parallelization is employed and each processor in the pool of parallel computing nodes performs the selection and the variation including crossover, mutation, and local search. This work employs multicore CPU with four cores, and four different UAV paths are optimized.

The work of Yang et al. [124] proposes a web-based parallel GA optimization framework based on a high-throughput distributed computation environment. A global parallelization of NSGA-II is implemented in a distributed computation environment by using the HTCondor computation environment. On this framework, the computation of the individual solution fitness value is executed as an HTCondor job. The optimization server is implemented with Node.js and JavaScript. Users can consume optimization services through the Internet by different browsers and facilities, such as Chrome, Internet Explorer, Firefox, and so on. Moreover, particular client pages can be constructed accounting to users' requirements.

Hong et al. [54] apply a PGA for data-mining task with fuzzy-based GA. They use the master-slave implementation where fitness evaluation of each fuzzy-rule with 10K transactions, which is the most time-consuming process.

Parallel cellular model. In making this survey, only one parallel cellular model has been found in the last six years out of the top 20 papers. The work of Pinel et al. [78] proposes a novel parallel design of cellular GA for GPU architecture, named as GraphCell. The population is initialized at random, while one solution is initialized by using the Min-min heuristic, a deterministic algorithm proposed by Ibarra and Kim in 1977. The recombination operator is executed with one thread per task of a solution; unlike the usual design, it evolves one solution

in every single GPU thread. This means each solution of the population is recombined in parallel. They use GPU holding 14 multi-processors of 32 cores each—in total 448 processors—and employ CUDA to implement the proposed method. The proposed method is tested on problem instances of the independent task assignment problem ranging from 512 tasks over 16 machines, to 65,536 tasks over 2,048 machines. The problem instances are randomly generated with the high task and machine heterogeneity. The population is set to 8×8, and the neighborhood used is von Neumann. The performance of the proposed method is clearly higher with respect to the CPU version when the problem size increases, especially a time gain ratio of ×538 for the biggest instance.

Parallel hybrid. Table 8 summarizes the characteristics of the parallel hybrid models used in the analyzed papers in Table 5. The "Hybrid" column indicates what kinds of implementation are combined, while the "Detail" column contains an explanation of an implementation.

The work of Porta et al. [80] considers three architectures of parallelism: multicore CPU based on shared memory with multi-threading, cluster based on message passing, and their combination. In the multicore CPU architecture, the global parallelization is used, where a genetic loop, selection, crossover, mutation, and evaluation is performed in each thread. However, in the cluster architecture, the island model is used, where each computing node generates a different initial population and executes the complete algorithm. After that, the best solutions obtained from executing different slave nodes join the new population on a master node, and slave nodes start their executions from the common joined new population. The hybrid architecture combines them. In particular, each slave node has a different population, and it executes the complete algorithm while performing the genetic loop consisting of selection, crossover, mutation, and replacement with a number of threads.

The work of Shayeghi et al. [96] proposes a pool-based Birmingham cluster genetic algorithm (Pool-BCGA), which is an extension of generation-based BCGA. The traditional generation-based BCGA, which combines local minimization with GA, is a sequential code where local optimization of individuals is not independent of each other. However, the proposed Pool-BCGA is a kind of asynchronous global parallelization model, in which one global population pool is managed by a master node, while slave nodes get solutions to be evolved from the global population pool and return execution result. Each GA run performs genetic operators such as mutation, crossover, and executes local minimization for a newly generated solution. The cluster environment with up to 240 processors is used and each pool subprocess of the proposed method is running on 48 processors.

The work of Roberge et al. [89] reports the application of a PGA implemented on GPU to optimize the switching angle of multilevel inverters. This paper employs the hybrid approach, the island model, and the evaluation parallelization. The purpose of this parallelization is to enable real-time control of multilevel inverters. For the evaluation parallelization, four strategies are taken into account to parallelize and accelerate the fitness evaluation on GPU. The proposed implementation achieved a time gain of 469×, and the execution times ranged from 38 to 164 ms, allowing for real-time control.

Clusters and GPUs are now popular hardware for parallelization, and they are being explored for larger scalability of algorithms. Their combination, at the hardware level and the algorithmic level, is making hybrid algorithms an interesting open line of research for the future.

4.2 Hardware

Hardware is one of the most important components to implement a PGA in practice. There are many kinds of hardware architecture to be employed to implement parallel systems. In Table 5, four hardware architectures are usually found: multicore CPU, cluster, cloud computing, and GPU.

Apart from these hardware platforms, other hardware architectures can be taken into account to implement a PGA; in particular, grid computing, mobile/smartphone, and FPGA.

Multicore CPU. Multicore CPU is a simple but powerful hardware architecture, on which two or more processing cores are embedded into one CPU. Usually, multiple cores share one memory space (shared memory) and can exchange data through it. In Table 5, seven works are found to use multicore CPUs. When multicore CPUs are used in PGAs, 3–8 processors are used for parallelization in the present literature. From the point of view of the PGA model, five works out of seven employ the parallel island model, while two works use general parallelization.

Cluster. A cluster architecture is constructed with two or more computers with single core or multicore CPUs and distributed memories. The computers in the cluster architecture are generally connected to each other through local area network (LAN) and/or wide area networks (WANs) and communicate by using message passing and/or socket communication. In the cluster architecture, the number of processors can be increased by enlarging the number of used computers, while it is (usually) impossible to increase the number of processors of a multicore CPU architecture. Actually, when the cluster architecture is used in the PGA works of Table 5, up to 240 processors are utilized. The work of Tosun et al. [110] utilizes an HPC system that consists of 46 computing nodes, each node has two CPUs, and each CPU has four cores, so a total of 368 processors are available on this system.

GPU. A graphics processing unit (GPU) can be used to accelerate real-time image processing, while it has recently attracted attention for general-purpose use; in particular, for evolutionary computation and deep neural networks. Unlike multicore CPU, GPUs have hundreds or thousands of processors in one chip. Three papers in Table 5 utilize GPUs with hundreds of cores, each having 512 cores [89], and 448 cores [78]. GPUs are applied to the hybrid model and the parallel cellular model. Other than these, some additional important papers of GPU-based PGAs were published in the early 2000s. For example, the work of Pospichal et al. [81] proposed an implementation of the island-model PGA that used 1,024 blocks and 256 threads for each block, achieving 7,000 times faster execution on the benchmark problem than the single threaded version. Also, the work of Kai and Zhen [117] used a GPU-based PGA to generate daily activity plans for 1K individuals and households, reporting 23–32 times faster execution than the one-CPU sequential implementation.

Cloud computing. Cloud computing is a paradigm for delivering compute power, database storage, applications, and other IT resources through a cloud services platform via the Internet in a very transparent manner. Enterprise cloud services are provided such as Amazon Web Services, Google Cloud Platform, and Microsoft Azure. Such services are certainly utilized for the use of parallel computing including PGA executions. Actually the work of Wen et al. [118], listed in Table 5, runs their PGA algorithm on 30 servers hosted on Amazon Web Services, while several works study the implementation of PGAs on cloud systems [71, 125].

Grid computing. Grid computing is a technology that aggregates resources such as distributed processors, data, storage systems, and networks and makes them available as one huge resource. One of the important aspects of grid computing is to gather surplus computing resources and make effective use of it in demand areas. In PGA domain, the grid computing technology has attracted attention; for example, Georgiev and Atanassov [46] and Oliveira et al. [113] implemented an island model PGA on grid computing environments in their works.

Mobile/Smartphone. In the last decade, smartphones and mobile devices have rapidly become widespread, and their processing capability has remarkably improved. For this reason, some researchers study to make use of smartphones and mobile devices as computing resources like cluster or grid computing architecture [23, 98].

FPGA. The field programmable gate array (FPGA) is a reconfigurable integrated circuit after manufacturing. To reconfigure an FPGA, the hardware description language (HDL) is generally

used. The main advantage of FPGA is its flexibility that enables for high efficiency and high-speed processing of specific applications by constructing logic circuit–specified to tasks. Another advantage is its high parallel performance. By constructing logic circuits for a certain process, it can be executed in parallel during one CPU clock. Utilizing these advantages, a PGA architecture using FPGA has been proposed in several studies [43, 52].

4.3 Software

Several APIs are found in the present literature to quickly and certainly implement PGA algorithms in practice. In Table 5, four APIs are found: Parallel Computing Toolbox, Distributed Computing Server, MPI, and CUDA.

Parallel Computing Toolbox. This API is used in two papers [14, 88] with MATLAB. Parallel Computing Toolbox is provided by MathWorks and supports the solution of computationally and data-intensive problems using multicore processors.

Distributed Computing Server. This API is also provided by MathWorks to support parallel computing on a distributed computation environment such as a cluster computing architecture. The work of Devos et al. [38] employs Distributed Computing Server with MATLAB on a cluster of machines.

MPI. This is a standard API for message passing on parallel computing architecture. MPI libraries such as OpenMPI and MPJ are used as major parallel API with Fortran, C++, and Java. Actually, eight works [48, 66, 80, 99, 110, 118, 126, 127] in Table 5 employ MPI, MPJ (Java implementation of MPI), and OpenMPI (one open source MPI implementation) on cluster computing architecture or cloud environment.

CUDA. Especially for GPU architectures, the CUDA toolkit is used. CUDA has been developed and provided by NVIDIA, and it is a platform for general purpose parallel computing on GPUs, so several C/C++ compiler and APIs are provided. On the PGA works using GPU in Table 5, two works [78, 89] employ CUDA on GPU developed by NVIDIA.

To implement PGAs, several EA libraries are utilized to quickly put the proposed algorithm in practice. Although most of the papers do not mention which library they use or they do not utilize any libraries (this is because the "Library" column in Table 5 is almost empty), a few papers in Table 5 employ libraries such as GEATbx, SiPESC.OPT, PGAPack, and jMetal.

GEATbx. GEATbx [79] is a toolbox for MATLAB. It does not only contain many optimization algorithms and problem instances and examples to run them, but also many helpful extensions, such as multi-objective optimization, constraint handling, problem-specific initialization and visualization, and multi-strategy and multi-population support. In Table 5, the work by Devos et al. [38] only utilized GEATbx.

SiPESC.OPT. SiPESC.OPT [123] is developed and provided by Dalian SiPESC CO., Ltd.[6]. SiPESC.OPT can be used to solve single or multi-objective, continuous or discrete design variables, and linear or nonlinear complex optimization problems by using a variety of algorithms. This also contains some analytical tools and a useful graphical user interface. In Table 5, the work of Yang et al. [124] utilized SiPESC.OPT.

PGAPack. PGAPack is a general-purpose, data-structure-neutral PGA library [64]. This is callable from Fortran or C and runs on uniprocessors, parallel computers, and workstation networks. Several data types are available, while it includes a large set of example problems and multiple choices for genetic operators. In Table 5, the work of Soufan et al. [99] only utilizes the PGAPack library.

[6] A web page written in Chinese.

Table 9. Academic and Generated Problem Instances in the Most Cited 20 Papers

Ref.	Instance	Domain	Data set
Wen et al. [118]	Problem generation	IoT applications orchestration	Randomly selected four types of orchestration graphs with 50, 100, 150, and 200 workflow nodes, 100 available IoT appliances as simulated agents, and randomly assigned security levels and network QoS levels
Kurdi [61]	Academic	Job shop scheduling	Standard JSSP benchmark instances by References [16, 44, 62]
Allahviranloo et al. [14]	Problem generation	Routing problem	Random fuzzy selective vehicle routing problem (FSVRP) consisting of 15 nodes and 2 vehicles and 200 nodes and 20 vehicles
Hong et al. [54]	Problem generation	Data mining	Random 10,000 purchase transactions with 64 items
Pinel et al. [78]	Problem generation	Task scheduling	Six different instance sizes, from 512 tasks × 16 machines 65,536 tasks × 2,048 machines, generated according to Reference [13]
Tosun et al. [110]	Academic	Quadratic Assignment Problem	QAPLIB [22]
Ghosn et al. [48]	Academic	Open-Shop Scheduling Problem	Taillard benchmark [104]
Zhang et al. [127]	Problem generation	Set cover problem	Different number of sensors and sensing radius
Dorronsoro et al. [39]	Problem generation	Task scheduling	512 tasks with 16 machines and 2,048 tasks with 64 machines, with two heterogeneities, generated according to Reference [13]

jMetal. The jMetal framework [40, 41] is implemented in the Java language and includes single-objective, multi-objective and parallel algorithms, problem instances, quality indicators, and variable representations. This framework is written as an object-oriented architecture so it enables users to quickly develop their own algorithms and solve their own optimization problem with state-of-the-art EA algorithms.

4.4 Academic and Generated Problem Instances

This section shows the details of academic and generated problem instances used in the most cited 20 PGA papers. Academic instances are very useful to compare the proposed PGA method versus other previous ones, because they can be compared under fair conditions. In addition, using problem generators is very useful, since the resulting instances can have any desired size and difficulty. This feature is suited to measure scalability or robustness of proposed PGA algorithms from the point of view of problem size.

Table 9 summarizes the characteristics of the instances of academic and problem generation. The "Instance" column indicates corresponding paper employs academic instance or generated problem instance, while the "Domain" column indicates the problem domain of instances used in papers. The "Data set" column indicates where problem instances come from or how they have been generated.

As benchmark instances, well-known NP-hard problems are employed in the PGA domain. Routing problems and task scheduling are usual test instances in PGA domain. The work of Allahviranloo et al. [14] applies a PGA to solve fuzzy selective vehicle routing problem (FSVRP) and they employ randomly generated small and large routing network for testing the proposed method.

Their work showed that the communication strategy occurring at every generation improved the search capability of the PGA at the cost of higher computation time.

The work of Wen et al. [118] provided an overview of fog-enabled orchestration for IoT services. In this paper, they demonstrated the initial results of orchestration for IoT services. Their experiment generated simulated data of randomly selected four types of orchestration graphs with 50, 100, 150, and 200 workflow nodes, 100 available IoT appliances, and randomly assigned security levels and network QoS levels. The experimental results demonstrated that a PGA is nearly 50% faster than a standard GA, while a PGA achieves at least 30% better solution than a standard GA.

Test instances of task scheduling are randomly generated for the desired number of tasks and a given number of tasks and given the number of machines. The scalability is then easy by changing these two values in the problem generator. In the work of Pinel et al. [78], six different instance sizes are tested—in particular, 512 tasks × 16 machines, 4,096 × 128; 8,192 × 256; 16,384 × 512; 32,768 × 1,024; and 65,536 × 2,048—while the work of Dorronsoro et al. [39] tests 512 tasks × 16 machines and 2,048 tasks × 64 machines with high and low heterogeneities of the tasks and machines. However, two kinds of research utilize benchmark instances provided by academic research. In the work of Kurdi [61], standard job shop scheduling benchmark instances provided by three works [16, 44, 62] are used, while Ghosn et al. [48] select Taillard benchmarks, which consist of a set of problem instances that are easy to generate. In their work, the number of machines and jobs is the same, and six different sizes (4, 5, 7, 10, 15, 20) are used.

One research by Tosun et al. [110] applies a PGA to Quadric Assignment Problems (QAPs), which is a well-known NP-hard combinatorial optimization problem. They tested their algorithm on QAPLIB [22], which includes more than 100 QAP standard benchmark instances, with problem sizes ranging from 12 up to 150 locations. The experiment in their work concluded that the increased number of processors allowed it to deal with large population and generations, which contributed to improve the quality of solutions.

In the work of Zhang et al. [127], the set cover problem of wireless sensor networks is solved, in which assignment of a number of sensors to different cover sets (groups of sensors) is optimized to enable each set to cover all target areas independently. They tested different number of sensors (5,000, 10,000, 15,000, 20,000, 25,000, and 30,000) with different sensing radius (5 and 8) and different number of cover sets.

4.5 Applications

This section is devoted to present the latest applications where PGAs have been instrumental to find relevant results, according to the fresh literature of the last six years in international journals. Algorithms of high efficiency and accuracy as PGAs are continuously being used in practice, and many authors just care about the results, not that much about the fundamentals of the algorithms, so the point of view in this section is accordingly on the applications themselves. Many researchers continuously apply PGA to solve problems in real-world applications. In particular, we found the following ones repeatedly in the literature reviewed for this survey: path finding, data mining, road traffic, land-use planning, nanoscience, electronics, building structure, and power systems. In this section, we discuss these applications, which were solved in PGA papers cited more than 20 times.

Table 10 shows a summary of applications and their problem characteristics in the most cited 20 papers. The "Application" column indicates the real-world task treated in the corresponding paper. The "# objectives" column indicates the number of objectives of the target mathematical model, while the "Fitness" column indicates whether the fitness value of the target problem is static or dynamic. The "Constraint" column indicates whether the target problem contains some constraints or not, while the "Solution" column indicates the representation of solution used by the solving PGA. The details of these studies related to real-world applications are discussed below.

Table 10. Applications in the Most Cited 20 Papers

Ref.	Application	# objectives	Fitness	Constraint	Solution
Roberge et al. [88]	Path finding	Mono	Static	No	Real-value
Devos et al. [38]	Data mining	Mono	Static	No	Real-value (binary coded)
Liu et al. [66]	Road traffic	Mono	Static	Yes	Real-value
Sahingoz [90]	Path finding	Mono	Static	No	Discrete
Porta et al. [80]	Land use planning	Mono	Static	No	Discrete
Shayeghi et al. [96]	Nanoscience	Mono	Static	No	Real-value
Roberge et al. [89]	Electronics	Mono	Static	No	Real-value
Yang et al. [124]	Building structure	Mono	Static	No	Mixed (Real-value and discrete)
Soufan et al. [99]	Data mining	Mono	Static	No	Binary
Ye and Huang [126]	Power system	Mono	Static	Yes	Real-value
Cao and Ye [26]	Land use planning	Mono	Static	No	Discrete

Path finding. The work of Roberge et al. [88] used PGA and parallel PSO to produce paths for unmanned aerial vehicles (UAVs). Each path is represented as a number of waypoints (3D coordinates) of UAV and is evaluated by penalizing path length, average altitude, the path through danger zones, required power of UAV, colliding with the ground, required fuel, and the path that cannot be smoothed. The island model parallelization is used, in which the migrations occur synchronously, 10 times throughout the optimization process, and new subpopulations are generated by randomly shufing a global population that joins all old subpopulations. The experiment is run on two quad-core CPUs (in total, eight cores). To implement the algorithm, the authors employ MATLAB and the Parallel Computing Toolbox. They achieved a quasi-linear speedup of 7.3 on eight cores and an execution time of 10 s for both algorithms, which shows the possibility of real-time path planning for UAV on a standard multi-core CPU. The work of Sahingoz [90] also solved the UAV path finding problem by applying a PGA. In this work, paths of multiple UAVs are optimized simultaneously. The global parallelization is employed and each processor of parallel computing nodes performs the selection, crossover, mutation, and local search. This work employed a multicore CPU with four cores, and four different UAV paths were optimized. Although not listed in Table 5, some other research tackled the UAV optimization. For example, Shaferman and Shima [95] applied coevolutional PGA to optimize a heterogeneous UAV team tracking a group of targets.

Data mining. The work of Devos et al. [38] applied PGA to data pre-processing and model selection of SVM classification. In particular, a solution consists of the two meta-parameters of SVM (C and G) and p pre-processing to be applied to the data. The global parallelism model was used here, and each slave node decodes the chromosome, applies the pre-processing to the dataset, constructs the SVM model with decoded meta-parameters, and evaluates this model by k-fold cross-validation. They used a High Performance Computing Cluster (Transtec AG, Tübingen) with one master computer and 16 slave nodes, which consist of four computing nodes using two dual-core processors. The proposed method was applied to the classification task, in particular, classifies whether olive oil comes from Liguria (a coastal region of north-western Italy) or other regions according to NIR or MIR spectra. The experimental result revealed that the SVM with pre-processing and parameter optimized by a PGA achieve higher classification accuracy than the previous classification methods. A combination of PGA with data mining or machine learning has also been studied by other researchers; for example, the work of Soufan et al. [99] used the global parallelization and developed a web-based tool, named as DWFS, for the feature selection in a variety of biomedical problems.

Road traffic. Liu et al. [66] applied PGA to traffic management; in particular, the practical speed-based toll design problem for the cordon-based Electronic Road Pricing (ERP) system in Singapore [66]. In this problem, the toll fare pattern for each charging link is optimized to maximize the total social benefit and average travel speed. Each solution is evaluated based on the probit-based stochastic user equilibrium problem with given toll charges, which needs much computational effort, because it executes the Monte Carlo simulation in each iteration. The global parallelization was chosen to evaluate each solution in parallel, and when 30 processors were used, the computation was accelerated by nearly 11 times, where the execution time decreased from around 30 hours to 2.7 hours. Since road-traffic optimization requires many computational efforts to simulate decision making of drivers or traffic congestion, PGA can be frequently found in the literature to speed up optimization for road-traffic problems. For example, the work of Potuzak [82] applied the global parallelization to road-traffic problems on the cluster architecture, and the same author extended it to reduce the frequency of communications of computing nodes [83]. The work of Shen et al. [97] proposed a GPU-based parallel NSGA-II for a road-traffic problem that minimizes the average delay and the average stop time simultaneously.

Land-use planning. The work of Porta et al. [80] used a PGA for land-use planning, in which each piece of land is allocated to the best category according to certain criteria and restrictions. They considered three implementations of PGA: the global, island, and their hybrid; and they did so on different architectures: multicore CPU based on shared memory with multi-threading, cluster based on message passing, and their hybrid. The work of Cao and Ye [26] also applied PGA—in particular, coarse-grained (island) PGA—to land-use planning. The full connected topology was used and their PGA was implemented on multicore CPU with four processors. The proposed model was applied to the land-use planning problem in Tongzhuo Newtown, Beijing, China.

Nanoscience. The work of Shayeghi et al. [96] applied Pool-BCGA to make global geometry optimization on a molecular cluster, in which local optimizations are the bottlenecks in a global optimization when using *ab initio* methods, which are computational chemistry methods based on quantum chemistry and used for global geometry optimization task of nanocluster structures. Five subprocesses of GAs simultaneously ran and each subprocess was executed on 48 processors; in total, 240 processors were used on a cluster environment. The proposed algorithm performed a linear scale-up while the traditional (sequential) BCGA gets stacked in plateaus when using a large number of cores. Similar to this, some works applied PGAs to an optimization in the nanoscience domain; for example, Ir_N ($N = 10–20$) clusters in Reference [33], MgO(100)-supported AuIr subnanoalloys in Reference [32], Ru-Pt binary nanoalloys in Reference [36], and MgO(100)-supported Pd, Au, and AuPd nanocluster structures in Reference [55].

Electronics. The work of Roberge et al. [89] reported the application of a PGA implemented on GPU to optimize the switching angle of multilevel inverters. The purpose of this parallelization was to enable the real-time control of multilevel inverters. This paper applies the hybrid approach that integrates the island model and the evaluation parallelization. For the parallel evaluation, four strategies were taken on the GPU. The proposed implementation achieved a speedup of 469× and the execution times ranged from 38 to 164 ms, allowing for real-time control.

Building structure. The work of Yang et al. [124] applied the global parallelized NSGA-II to the building energy consumption optimization, which uses the simulation model of the KUBIK building, located in Bilbao, Spain, and developed in EnergyPlus, to evaluate fitness value. The experiment used 48 computing nodes arranged as eight computers with a 12-core processor. The parallel optimization framework with 48 processors achieved a speedup ratio of 39.3. Other than this, the work of Park et al. [77] proposed a distributed NSGA-II to a seismic retrofit design of a 2D steel frame structure and 3D irregular reinforced concrete structure. Since much evaluation

time was needed to calculate seismic performances of a retrofitting design, they ran the global parallelization to evaluate solutions in parallel.

Power systems. The work of Ye and Hung [126] applied a parallel NSGA-II (PNSGA-II) to solve transient stability constrained optimal power flow (TSCOPF) in power system operation. In particular, this work incorporated an additional objective, transient stability, into the classical model, which originally had one objective: the total cost of fuel consumed by a generator. Since the proposed multi-objective TSCOPF problem needs much computational time, especially for large-scale power system and multiple contingencies, this paper adopted a parallel approach to shorten the execution time. As target problems, they used the IEEE 39, 300, and 678 bus systems as benchmarks and compared a parallel NSGA-II with the traditional TSCOPF solver (the interior point method (IPM)). Their case study demonstrated that the proposed PNSGA-II can achieve a theoretically strictly transient stable solution while reducing the optimization time. Such speedup enables their proposed method to be applied to realistic large-scale or multi-contingency power systems, where the traditional IPM cannot be applied.

5 CHALLENGES AND NEW TRENDS

This section contains a brief summary of open challenges for the further development of PGAs, which could be used as a set of research lines to foster in new master and PhD theses. Also, we include information on already developing technologies and fields related to PGAs, which could be used to guide interested readers to domains new to them. We then make a summary of what is worth to be addressed and what is worth to know because it is happening as of today, hopefully making here a case of discussion and thinking for the PGA community.

We must notice that this is a reflection based on the knowledge of the authors of this article, and then a topic for open debate, of course. Indeed, we will merge challenges and future trends in the next list of items, since they both interplay in complex manners. Let us start.

Few seconds execution. In real-time applications like in critical systems (Industry 4.0, drivers, cars, planes…) a fast execution is mandatory [18, 63]. PGAs running completely within a few seconds (or milliseconds) is a challenge for present research. For this, we should probably need more than fast parallel hardware, and then we should go for theory (to reduce the number of function evaluations) and specialized operators.

Beating the state-of-the-art. PGAs are often just used to reduce the running time, and authors base their contribution in this reduction. But PGAs allow a much larger power than this, as they could lead to a reduction in the number of fitness function calls and produce an increment in the precision of the computed solution as well. New PGA models and implementations are wanted to achieve this *parallel* maximization of benefits and reduction of costs [3].

Scalability. One main concern in present research is how narrow are the studies published in impact journals. In particular, authors often only address one problem instance, or a few that share a similar dimensionality. The main challenge is to face PGAs to many problem instances where all of them are of very different dimensions. For example, go for thousands and millions of variables when solving SAT [120], or for hundreds, thousands, and millions of cities when solving TSP [35, 50]. If the proposed algorithm shows quasi-linear scalability (growth of the effort of the same algorithm when facing increasingly large problems), then this technique is worth to publish and learn [12]. The number of new techniques is so large nowadays, and the need for scalability is so normal in the real world, that we should only care about algorithms showing with numerical data that they scale appropriately.

Robustness. This is an often-mentioned issue in most search, optimization, and learning studies at present, though not so often well defined. We can understand robustness as the *persistance of good results* shown by an algorithm under changes (of different intensity) in its parameters [53,

121]. Having PGAs that show robustness in this sense will then make reliable tools for final applications in the real world. The feature of metaheuristics consisting in giving a different answer to the same problem in different runs (non-determinism) sometimes is not welcomed by experts, who want a reliable and even the same result always. At least, considering the probability of distributions instead of scalars (e.g., on TSP, consider that travel times are not scalar, but normal distributions with an average and a standard deviation) and reporting on confidence intervals for algorithms is a demanded feature of modern solvers that researchers should be aware of [6, 31]. Robustness is often also understood as algorithms dealing with uncertainties in the data, or even dealing with dynamic environments [8], where the same individual can get several different fitness values along the search done in the algorithm, clearly defining new topics for research.

Multiobjective. The combination of PGAs and MO studies/concepts is now a classic in literature [57, 68]. The checking of constraints in complex MO problems takes a lot of time, and PGAs are a way to go there. In many MO problems, checking constraints requires more time than evaluating the different objective functions. In other cases, computing hypervolume can be eased by using a PGA. Still in other situations, new models for solving MO problems can be built by decentralizing the MO algorithm, thus having more efficient algorithms that could construct better Pareto fronts in diversity and quality. Also, computing more solutions in the front can come as a benefit of using a PGA. Finally, as to multicriteria decision making (MCDM), we still have to see interactive PGAs dealing with one or more users stating their preferences, either offline or online in an interactive way to allow new applications in design (e.g., designing a building, creating a painting…) and market (e.g., analyzing bank data, determining the best medical treatment for a patient…).

Interactive/online. This topic is still to rise, but it is clear that if users need a real-time answer to their queries for an optimal solution to a complex real problem, then PGAs are the way to go [76]. First, modern hardware would be needed (GPUs mainly, but also FPGAs or cloud computing), and thus the importance of building algorithms for new platforms. Second, graphical user interfaces (and vocal ones, and others) would have a say here. We also link now to very efficient execution (seconds or milliseconds) and user preferences, targeting a new body of services where a PGA is at the heart of the intelligence offered to the user. These systems sometimes need to wait for the user and then go for a quick answer to the defined problem in a kind of burst behavior that opens new avenues on how all this should be managed.

Body of knowledge. The difficulty of having an explicit body of knowledge in PGAs is the main problem for its development as a research field. Researchers have a hard time finding relevant papers, there are no clear benchmarks to evaluate the quality of PGAs, and the list of best practices is still weak and scattered over different journals and conferences. Often, researchers interested in PGAs only read and publish in journals on GAs, but not on parallelism. The contrary also happens, since there is no such journal or conference with yearly execution to find data on this domain. To know more on the body of knowledge, researchers should at least consult the previous surveys on this topic [2, 3, 7, 11, 24, 56, 60, 106, 109, 112]. We hope to make a contribution to this body of knowledge with this article, but obviously it should happen at the level of the interested community, and community is still sparse, since most researchers are going for applications of PGAs and not for their fundamentals. That is a must: Going for the working principles and bases for the construction and analysis of PGAS themselves is a real future challenge.

Trade-off usability/efficiency. PGAs are often used, because researchers want to build a complex solver that is efficient at the same time. Sometimes, making the algorithm usable means to forget about the internals of the algorithm and focus on the user interface, full of data relevant for the application domain. And thus, it is normal that efficiency and even science are sacrificed for the sake of the nice application at hand. We here warn that PGAs, and metaheuristics in

general, are not magic recipes for solving any problem. They are not black boxes amenable for users to build their applications while forgetting how the underlying algorithm is working. It is very normal that these black boxes make researchers happy at first when using PGAs but soon later a wall is faced, where no advances are possible without going into the algorithm operations. In particular, efficiency is a must, even if we talk of a few seconds of execution, since in this domain we care also about small times, because they usually reveal themselves as important when growing in dimension. In fact, an analysis of the algorithm is always a good idea, to be able of further enhancements [25, 65, 85, 114]. Not being able of considering this tradeoff (usability/efficiency) is the cause for stagnation in the research with new software parallel tools, that easily go inefficient on larger problems just because researchers do not want (have any time) to enter the guts of the PGA used.

Big data+parallel. Although no one has a clear idea of what big data is, the fancy term got known after several initial attempts to use parallel platforms without caring for parallel details. Big data researchers use software such as Hadoop and Spark as an abstract layer to use parallelism to harvest gigabytes of information to offer a statistical or intelligent conclusion. Since map/reduce is the main paradigm here [25, 103, 122], found in MPI [92] and other parallel software and systems previous to the existence of big data, much can be said on how to configure and program on big data to efficiently use the underlying computing services. If all big data applications use Spark (or similar), and if Spark (and similar) software tools are a way of using map/reduce parallelism (and thus, to forget on the internal parallel platform), then big data is actually concerned by parallel issues, and the related algorithms, and metrics, and body of knowledge. So, we encourage researchers interested in big data to first understand the kind of algorithms that can be run in parallel in a large number of computers [3] so their solutions actually exploit the hardware and can go for big data, bit volatility big variability, and the rest of "v" found in this domain [86].

Edge/Fog computing. A fact in research is that we proceed in waves of interest. And a new wave of interest is starting on not going centralized (as big data implies) for the analysis of data. Fog (or edge) computing [1, 15, 84, 87] claims that the computing should be done in the edge; that is, in the devices near the input of data to the system. This means running algorithms on smartphones, routers, and some other intermediate devices, no longer using a set of computers as the central location for data processing and analysis. Then building new PGAs running on portable devices (tablets, smartphones) [29, 72], small computing–limited devices (Raspberry Pi 3, Arduino, wearables), and other edge devices (routers, like the ones from Cisco running iOX) [73] is a major challenge and a future trend that we will see happen in the coming years. The ones arriving quickly there will be well positioned for the innovation, research, and development of the next generation of algorithms and services.

Small devices. After making clear that it is important to open new lines of research concerning small devices, one wonders about what are the challenges and interesting venues here. First, we can mention the obvious need to run complex algorithms with a slow set of cores (compared to desktop machines), a short memory, and with a low impact in battery consumption [6, 30, 67, 72]. Algorithms should then focus on theoretical results to perform an efficient search [28], but also new PGA models could arise from using many cores in a small device and linking the algorithm between different devices at the same time. Second, we need algorithms to be aware of the hardware [75, 91], and this means they perform operations that change in time depending on the availability of computing resources. This is still not well known: How mutation, crossover, and local techniques should perform a meaningful numerical search while at the same time they adapt to a volatile set of small devices and running/communication errors along the life of the algorithm? How to build algorithms here to deal with thousands of devices and to address problems in a crowd (voluntary) kind of computing is a wide and interesting line of research that will make PGAs and AI ubiquitous.

Exascale computing. This refers to computing systems capable of at least one exaFLOPS, or a billion calculations per second (10^{18} floating point operations per second or FLOPS) [17, 21, 34]. Arriving to an exascale computing power is a main scientific target for High Performance Computing (HPC) nowadays, and research agencies are funding this technology and the applications running on it. Exascale platforms will allow the development of new type of PGAs for very complex problems, probably hybrid techniques in the operations that they perform (exact plus approximate algorithms), combined synchronous/asynchronous communication policies, and log/short memory algorithms that are mixing results computed in different moments of the time because of the different speeds of the connected processing elements. In exascale computing, major challenges remain open, like using more than 20% of the computing power for a given algorithm and problem: Is this possible when we deal with PGAs? This example challenge also raises other questions, like the possibility of using PGAs in the core scheduling algorithm of the exascale platform so it is used full time. Thus, PGAs can be developed over these platforms, but at the same time, they can be used to develop such platforms: a new field of research.

Web services. In most research domains linked to optimization, search, and learning, researchers offer numerical results, data files with benchmarks, documents, and open software in the form of libraries. However, there are more ways to use the knowledge we all produce when researching, and building web services is one of them [93]. Indeed, an interesting one, since we could offer new optimization services for users through web services that are internally running PGAs on a local network (or multicore) [9]. That would be well accepted by practitioners, who will submit their problems (through a given description language, or XML file, or just source code in Java) and wait for an answer. Building microservices and merging them into more complex solutions would be of interest in the future, especially for a cohort of researchers who want to use well-tested and validated algorithms without going into their internals. This points to a new kind of market of specialized solvers used through reliable web services, thus representing an interesting shared domain for researchers and companies.

6 CONCLUSIONS

We now end this article by summarizing the main contents and findings in it. Let us first remind of the intention of not only having a source of information for modern research in PGAs, but in including the added value in every section so it is actually useful to a wide audience. We did our survey in a systematic way and defined four meaningful research questions that guided our discussion.

On the contents, we tried to visit all common places and then talked on models for PGAs, existing fresh works on this domain, and a guiding tour around the venues where PGAs are being endorsed. On conferences, *ACM GECCO* and *IEEE CEC* are the preferred places for publishing in PGAs. On journals, the two first ones have a clear profile in parallelism, *Concurrency and Computation: Practice and Experience* and *International Journal of Distributed Sensor Networks*, an explicit lesson that the "P" in PGAs has deep importance.

As a global conclusion, it is clear that the domain is healthy (almost 4K researchers publishing in the last years), though there is a trend in focusing in the applications and not that much in the algorithms themselves. This is good in the short term but not so good in the long term if we want to understand and better exploit these techniques for future practice. We really encourage researchers to go on the fundamentals and understand them well before passing to focus in the application itself to avoid the classic "wall" of no further enhancements in the performance for a lack of knowledge on the solver used.

The environment in research on PGAs is shifting in topics and places also. China is emerging as a source of many new works, and our understanding of PGAs would need to deal with a whole new

taxonomy on hardware and software that is somewhat complex because of the many advances in these two domains. Researchers and practitioners should pay attention to the fundamentals of the models, the best practices in non-deterministic algorithms, and the basic concepts existing in the domain of parallelism and HPC that are, to some degree, continuously missed in present works where the application or at most the algorithm is the target, while a whole body of information exists for parallel processing that could lead this domain to new levels of creative thinking and highly efficient solvers.

The literature in the recent six years was reviewed from the point of view of PGA models, hardware, software, academic and generated problem instances, and real-world applications. In the recent literature, all PGA models, the global parallelization model, the parallel island model, the cellular model, and the hybrid model, were used. The multicore CPU with several processors and the cluster architecture with tens or hundreds of processors on many computing nodes were popular in the recent literature. However, GPU with thousands of cores is raising in the literature, especially linked to the use of CUDA, while cloud systems such as Amazon Web Services were also utilized. To implement PGAs, several APIs such as Parallel Computing Toolbox, Distributed Computing Server, MPI, and CUDA were frequently utilized, while useful libraries also helping authors to easily implement PGAs, for example, GEATbx, SiPESC.OPT, PGAPack, and jMetal. Academic and generated problem instances were utilized to test PGA methods; in particular, they are useful to investigate the scalability of PGA methods, because such academic and generated problem instances can be easily scaled up by increasing the number of design variables. Many researchers paid attention to just the real-world application of PGAs. They applied PGAs to several domains, such as path finding, data mining, road traffic, land planning, nanoscience, electronics, building structure, and power system. Many of them are computationally expensive or have to be solved in short computing times, like the ones found in real-time critical systems.

We hope that our tables and figures represent a quick means to understand this domain. We also tried to contribute in giving details as much as possible but without getting lost because of them. A global picture of the domain is a nice lesson that this survey can help grasp. At some moments, we went into details on new topics and open challenges so new PhD theses can have a starting point based on the many suggestions across this article.

This survey represents a big structured and explained database of recent work and also a long list of suggestions and data and tools that will hopefully help others in making, not only research, but development and even innovation to transfer efficient solvers to companies. Of course, more work still has to be done in defining the fundamentals of PGAs and in developing scientific thinking in the future advances in this field. We do believe PGAs are precious tools that could help anyone out there to improve his/her techniques, thus solving open problems and even leading to funny and intuitive ways of addressing complex problems.

REFERENCES

[1] Cisco Systems. 2015. *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are*. White paper. Cisco Systems, Inc., San Jose, CA.

[2] Panagiotis Adamidis. 1994. *Parallel Evolutionary Algorithms: A Review*. Technical Report. University of Thessaloniki.

[3] Enrique Alba. 2005. *Parallel Metaheuristics: A New Class of Algorithms*. Vol. 47. John Wiley & Sons.

[4] Enrique Alba, Christian Blum, Pedro Isasi, Coromoto León, and Juan Antonio Gómez (Eds.). 2009. *Optimization Techniques for Solving Complex Problems*. John Wiley & Sons, Inc. DOI: https://doi.org/10.1002/9780470411353

[5] Enrique Alba, Christian Blum, Pedro Isasi, Coromoto León, and Juan Antonio Gómez (Eds.). 2009. *Optimization Techniques for Solving Complex Problems*. Wiley.

[6] Enrique Alba, Francisco Chicano, and Gabriel Luque (Eds.). 2017. Smart City. In *Proceedings of the 2nd International Smart Cities Conference (Smart-CT'17)*. Lecture Notes in Computer Science, Vol. 10268. Springer. DOI: https://doi.org/10.1007/978-3-319-59513-9

[7] Enrique Alba, Gabriel Luque, and Sergio Nesmachnow. 2013. Parallel metaheuristics: Recent advances and new trends. *Int. Trans. Oper. Res.* 20, 1 (2013), 1–48. DOI: https://doi.org/10.1111/j.1475-3995.2012.00862.x

[8] Enrique Alba, Amir Nakib, and Patrick Siarry. 2013. *Metaheuristics for Dynamic Optimization.* Vol. 433. Springer.

[9] E. Alba and J. G. Nieto. 2005. *ROS (Remote Optimization Service).* Technical Report Informe Técnico ITI 05-08, Dpto. de Lenguajes y CC.CC. Universidad de Málaga. (in Spanish).

[10] E. Alba and M. Tomassini. 2002. Parallelism and evolutionary algorithms. *IEEE Trans. Evol. Comput.* 6, 5 (Oct. 2002), 443–462. DOI: https://doi.org/10.1109/TEVC.2002.800880

[11] Enrique Alba and José M. Troya. 1999. A survey of parallel distributed genetic algorithms. *Complexity* 4, 4 (March 1999), 31–52. DOI: https://doi.org/10.1002/(SICI)1099-0526(199903/04)4:4⟨31::AID-CPLX5⟩3.3.CO;2-W

[12] Enrique Alba and José M. Troya. 2002. Improving flexibility and efficiency by adding parallelism to genetic algorithms. *Statist. Comput.* 12, 2 (01 Apr. 2002), 91–114. DOI: https://doi.org/10.1023/A:1014803900897

[13] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Debra Hensgen, and Sahra Sedigh. 2000. Representing task and machine heterogeneities for heterogeneous computing systems. *Tamkang J. Sci. Eng.* 3, 3 (2000), 195–207.

[14] Mahdieh Allahviranloo, Joseph Y. J. Chow, and Will W. Recker. 2014. Selective vehicle routing problems under uncertainty without recourse. *Transport. Res. Part E: Logist. Transport. Rev.* 62 (2014), 68–88. DOI: https://doi.org/10.1016/j.tre.2013.12.004

[15] Muhammad Rizwan Anawar, Shangguang Wang, Muhammad Azam Zia, Ahmer Khan Jadoon, Umair Akram, and Salman Raza. 2018. Fog computing: An overview of big IoT data analytics. *Wirel. Commun. Mobile Comput.* 2018 (2018).

[16] David Applegate and William Cook. 1991. A computational study of the job-shop scheduling problem. *ORSA J. Comput.* 3, 2 (1991), 149–156. DOI: https://doi.org/10.1287/ijoc.3.2.149

[17] Muhammad Usman Ashraf, Fathy Alburaei Eassa, Aiiad Ahmad Albeshri, and Abdullah Algarni. 2018. Toward exascale computing systems: An energy efficient massive parallel computational model. *Int. J. Adv. Comput. Sci. Applic.* 9, 2 (2018), 118–126. DOI: https://doi.org/10.14569/IJACSA.2018.090217

[18] Radhakisan Baheti and Helen Gill. 2011. Cyber-physical systems. *Impact Contr. Technol.* 12, 1 (2011), 161–166.

[19] Richard J. Bauer. 1994. *Genetic Algorithms and Investment Strategies.* Wiley.

[20] Christian Blum and Andrea Roli. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.* 35, 3 (2003), 268–308.

[21] Katherine Bourzac. 2017. Supercomputing poised for a massive speed boost. *Nature* 551 (11 2017), 554–556.

[22] Rainer E. Burkard, Stefan E. Karisch, and Franz Rendl. 1997. QAPLIB—A quadratic assignment problem library. *J. Global Optim.* 10, 4 (01 June 1997), 391–403. DOI: https://doi.org/10.1023/A:1008293323270

[23] F. Büsching, S. Schildt, and L. Wolf. 2012. DroidCluster: Towards smartphone cluster computing—The streets are paved with potential computer clusters. In *Proceedings of the 32nd International Conference on Distributed Computer Systems Workshops.* 114–117. DOI: https://doi.org/10.1109/ICDCSW.2012.59

[24] E. Cantú-Paz. 1998. A survey of parallel genetic algorithms. *Calc. Parall.* 10 (1998).

[25] E. Cantú-Paz and D. E. Goldberg. 1999. On the scalability of parallel genetic algorithms. *Evol. Comput.* 7, 4 (1999), 429–449.

[26] Kai Cao and Xinyue Ye. 2013. Coarse-grained parallel genetic algorithm applied to a vector based land use allocation optimization problem: The case study of Tongzhou Newtown, Beijing, China. *Stoch. Environ. Res. Risk Assess.* 27, 5 (01 July 2013), 1133–1142. DOI: https://doi.org/10.1007/s00477-012-0649-y

[27] H. R. Cheshmehgaz, H. Haron, and A. Sharifi. 2015. The review of multiple evolutionary searches and multi-objective evolutionary algorithms. *Artif. Intell. Rev.* 43, 3 (01 Mar. 2015), 311–343. DOI: https://doi.org/10.1007/s10462-012-9378-3

[28] Francisco Chicano, L. Darrell Whitley, and Enrique Alba. 2011. A methodology to find the elementary landscape decomposition of combinatorial optimization problems. *Evol. Comput.* 19, 4 (2011), 597–637. DOI: https://doi.org/10.1162/EVCO_a_00039

[29] Christian Cintrano and Enrique Alba. 2016. Genetic algorithms running into portable devices: A first approach. In *Advances in Artificial Intelligence*, Oscar Luaces, José A. Gámez, Edurne Barrenechea, Alicia Troncoso, Mikel Galar, Héctor Quintián, and Emilio Corchado (Eds.). Springer International Publishing, Cham, 383–393.

[30] Christian Cintrano and Enrique Alba. 2016. Genetic algorithms running into portable devices: A first approach. In *Proceedings of the 17th Conference of the Spanish Association for Artificial Intelligence—Advances in Artificial Intelligence (CAEPIA'16).* 383–393. DOI: https://doi.org/10.1007/978-3-319-44636-3_36

[31] Christian Cintrano, Francisco Chicano, and Enrique Alba. 2017. Robust bi-objective shortest path problem in real road networks. In *Proceedings of the 2nd International Smart Cities Conference (Smart-CT'17).* 128–136. DOI: https://doi.org/10.1007/978-3-319-59513-9_13

[32] Jack B. A. Davis, Sarah L. Horswell, and Roy L. Johnston. 2016. Application of a parallel genetic algorithm to the global optimization of gas-phase and supported gold–iridium sub-nanoalloys. *J. Phys. Chem. C* 120, 7 (02 2016), 3759–3765. DOI: https://doi.org/10.1021/acs.jpcc.5b10226

[33] Jack B. A. Davis, Armin Shayeghi, Sarah L. Horswell, and Roy L. Johnston. 2015. The Birmingham parallel genetic algorithm and its application to the direct DFT global optimisation of IrN (N = 10-20) clusters. *Nanoscale* 7, 33 (2015), 14032–14038. DOI: https://doi.org/10.1039/C5NR03774C

[34] Koen De Bosschere. 2012. *Applications, Tools and Techniques on the Road to Exascale Computing*. Vol. 22. IOS Press.

[35] Kalyanmoy Deb and Christie Myburgh. 2016. Breaking the billion-variable barrier in real-world optimization using a customized evolutionary algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'16)*. ACM, New York, NY, 653–660. DOI: https://doi.org/10.1145/2908812.2908952

[36] Ilker Demiroglu, Kezi Yao, Heider A. Hussein, and Roy L. Johnston. 2017. DFT global optimization of gas-phase subnanometer ru–pt clusters. *J. Phy. Chem. C* 121, 20 (05 2017), 10773–10780. DOI: https://doi.org/10.1021/acs.jpcc.6b11329

[37] James Devillers (Ed.). 1996. *Genetic Algorithms in Molecular Modeling*. Academic Press, London. DOI: https://doi.org/10.1016/B978-012213810-2/50005-0

[38] Olivier Devos, Gerard Downey, and Ludovic Duponchel. 2014. Simultaneous data pre-processing and SVM classification model selection based on a parallel genetic algorithm applied to spectroscopic data of olive oils. *Food Chem.* 148 (2014), 124–130. DOI: https://doi.org/10.1016/j.foodchem.2013.10.020

[39] Bernabé Dorronsoro, Grégoire Danoy, Antonio J. Nebro, and Pascal Bouvry. 2013. Achieving super-linear performance in parallel multi-objective evolutionary algorithms by means of cooperative coevolution. *Comput. Oper. Res.* 40, 6 (2013), 1552–1563. DOI: https://doi.org/10.1016/j.cor.2011.11.014

[40] Juan J. Durillo and Antonio J. Nebro. 2011. jMetal: A Java framework for multi-objective optimization. *Adv. Eng. Softw.* 42 (2011), 760–771. DOI: https://doi.org/10.1016/j.advengsoft.2011.05.014

[41] Juan J. Durillo, Antonio J. Nebro, and Enrique Alba. 2010. The jMetal framework for multi-objective optimization: Design and architecture. In *Proceedings of the IEEE Congress on Evolutionary Computation*. 1–8. DOI: https://doi.org/10.1109/CEC.2010.5586354

[42] W. Fan, Y. Liu, B. Tang, F. Wu, and H. Zhang. 2016. Exploiting joint computation offloading and data caching to enhance mobile terminal performance. In *Proceedings of the IEEE Globecom Workshops (GC Wkshps'16)*. 1–6. DOI: https://doi.org/10.1109/GLOCOMW.2016.7848902

[43] Rasoul Faraji and Hamid Reza Naji. 2014. An efficient crossover architecture for hardware parallel implementation of genetic algorithm. *Neurocomputing* 128 (2014), 316–327. DOI: https://doi.org/10.1016/j.neucom.2013.08.035

[44] H. Fisher and G. L. Thompson. 1963. Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial Scheduling*, J. F. Muth and G. L. Thompson (Eds.). Prentice Hall, 225–251.

[45] Pablo García-Sánchez, Gustavo Romero, Jesús González, Antonio Miguel Mora, Maribel García Arenas, Pedro Ángel Castillo Valdivieso, Carlos M. Fernandes, and Juan Julián Merelo Guervós. 2016. Studying the effect of population size in distributed evolutionary algorithms on heterogeneous clusters. *Appl. Soft Comput.* 38 (2016), 530–547. DOI: https://doi.org/10.1016/j.asoc.2015.09.052

[46] D. Georgiev and E. Atanassov. 2014. Extensible framework for execution of distributed genetic algorithms on grid clusters. In *Proceedings of the 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO'14)*. 301–306. DOI: https://doi.org/10.1109/MIPRO.2014.6859581

[47] Ali Ghaheri, Saeed Shoar, Mohammad Naderan, and Sayed Shahabuddin Hoseini. 2015. The applications of genetic algorithms in medicine. *Oman Med. J.* 30, 6 (11 2015), 406–416. DOI: https://doi.org/10.5001/omj.2015.82

[48] Steve Bou Ghosn, Fouad Drouby, and Haidar M. Harmanani. 2016. A parallel genetic algorithm for the open-shop scheduling problem using deterministic and random moves. *Int. J. Artif. Intell* 14, 1 (2016), 130–144.

[49] M. Giacobini, M. Tomassini, A. G. B. Tettamanzi, and E. Alba. 2005. Selection intensity in cellular evolutionary algorithms for regular lattices. *IEEE Trans. Evol. Comput.* 9, 5 (Oct. 2005), 489–505. DOI: https://doi.org/10.1109/TEVC.2005.850298

[50] David E. Goldberg, Kumara Sastry, and Xavier Llorà. 2007. Toward routine billion-variable optimization using genetic algorithms. *Complexity* 12, 3 (2007), 27–29. DOI: https://doi.org/10.1002/cplx.20168

[51] Y. Gong and A. Fukunaga. 2011. Distributed island-model genetic algorithms using heterogeneous parameter settings. In *Proceedings of the IEEE Congress of Evolutionary Computation (CEC'11)*. 820–827. DOI: https://doi.org/10.1109/CEC.2011.5949703

[52] L. Guo, D. B. Thomas, Ce Guo, and W. Luk. 2014. Automated framework for FPGA-based parallel genetic algorithms. In *Proceedings of the 24th International Conference on Field Programmable Logic and Applications (FPL'14)*. 1–7. DOI: https://doi.org/10.1109/FPL.2014.6927501

[53] Chung-Chien Hong. 2014. A robust method for designing the parameters of genetic algorithm. *Int. J. Oper. Res.* 11 (2014), 051–063.

[54] Tzung-Pei Hong, Yeong-Chyi Lee, and Min-Thai Wu. 2014. An effective parallel approach for genetic-fuzzy data mining. *Exp. Syst. Applic.* 41, 2 (2014), 655–662. DOI: https://doi.org/10.1016/j.eswa.2013.07.090

[55] Heider A. Hussein, Jack B. A. Davis, and Roy L. Johnston. 2016. DFT global optimisation of gas-phase and MgO-supported sub-nanometre AuPd clusters. *Phys. Chem. Chem. Phys.* 18 (2016), 26133–26143. Issue 37. DOI: https://doi.org/10.1039/C6CP03958H

[56] Fauzi Mohd Johar, Farah Ayuni Azmin, Mohamad Kadim Suaidi, A. S. Shibghatullah, Badrul Hisham Ahmad, Siti Nadzirah Salleh, Mohamad Zoinol Abidin Abd Aziz, and M. M. Shukor. 2013. A review of genetic algorithms and parallel genetic algorithms on graphics processing unit (GPU). In *Proceedings of the IEEE International Conference on Control System, Computing and Engineering*. 264–269. DOI: https://doi.org/10.1109/ICCSCE.2013.6719971

[57] Janusz Kacprzyk and Witold Pedrycz (Eds.). 2015. *Springer Handbook of Computational Intelligence*. Springer. DOI: https://doi.org/10.1007/978-3-662-43505-2

[58] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Technical Report. EBSE-2007-01. Keele University and Durham University.

[59] Barbara Kitchenham, Rialette Pretorius, David Budgen, O. Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen Linkman. 2010. Systematic literature reviews in software engineering—A tertiary study. *Inf. Softw. Technol.* 52, 8 (2010), 792–805. DOI: https://doi.org/10.1016/j.infsof.2010.03.006

[60] D. S. Knysh and V. M. Kureichik. 2010. Parallel genetic algorithms: A survey and problem state of the art. *J. Comput. Syst. Sci. Int.* 49, 4 (01 Aug. 2010), 579–589. DOI: https://doi.org/10.1134/S1064230710040088

[61] Mohamed Kurdi. 2016. An effective new island model genetic algorithm for job shop scheduling problem. *Comput. Oper. Res.* 67 (2016), 132–142. DOI: https://doi.org/10.1016/j.cor.2015.10.005

[62] S. Lawrence. 1984. *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*. Graduate School of Industrial Administration, Carnegie-Mellon University.

[63] Jay Lee, Behrad Bagheri, and Hung-An Kao. 2015. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manuf. Lett.* 3 (2015), 18–23. DOI: https://doi.org/10.1016/j.mfglet.2014.12.001

[64] D. Levine. 1996. Users guide to the PGAPack parallel genetic algorithm library. *Argonne Nat. Lab.* 9700, S 8703941 (1996).

[65] Yan Y. Liu and Shaowen Wang. 2015. A scalable parallel genetic algorithm for the Generalized Assignment Problem. *Parallel Comput.* 46 (2015), 98–119. DOI: https://doi.org/10.1016/j.parco.2014.04.008

[66] Zhiyuan Liu, Qiang Meng, and Shuaian Wang. 2013. Speed-based toll design for cordon-based congestion pricing scheme. *Transport. Res. Part C: Emerg. Technol.* 31 (2013), 83–98. DOI: https://doi.org/10.1016/j.trc.2013.02.012

[67] Oscar Luaces, José A. Gámez, Edurne Barrenechea, Alicia Troncoso, Mikel Galar, Héctor Quintián, and Emilio Corchado (Eds.). 2016. In *Proceedings of the 17th Conference of the Spanish Association for Artificial Intelligence—Advances in Artificial Intelligence (CAEPIA'16)*. Lecture Notes in Computer Science, Vol. 9868. Springer. DOI: https://doi.org/10.1007/978-3-319-44636-3

[68] Francisco Luna and Enrique Alba. 2015. Parallel multiobjective evolutionary algorithms. In *Springer Handbook of Computational Intelligence*. 1017–1031. DOI: https://doi.org/10.1007/978-3-662-43505-2_50

[69] M. D. McKay, R. J. Beckman, and W. J. Conover. 1979. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* 21, 2 (1979), 239–245. Retrieved from http://www.jstor.org/stable/1268522.

[70] Jean-Yves Potvin Michel Gendreau (Ed.). 2010. *Handbook of Metaheuristics*. Springer US.

[71] Fereydoun Farrahi Moghaddam, Reza Farrahi Moghaddam, and Mohamed Cheriet. 2015. Carbon-aware distributed cloud: Multi-level grouping genetic algorithm. *Cluster Comput.* 18, 1 (01 Mar. 2015), 477–491. DOI: https://doi.org/10.1007/s10586-014-0359-y

[72] J. Á. Morell and E. Alba. 2017. Distributed genetic algorithms on portable devices for smart cities. In *Smart Cities*, E. Alba, F. Chicano, and G. Luque (Eds.). Springer International Publishing, Cham, 51–62.

[73] J. Á. Morell and E. Alba. 2018. Running genetic algorithms in the edge: A first analysis. In *Advances in Artificial Intelligence*, F. Herrera, S. Damas, R. Montes, S. Alonso, Ó. Cordón, A. González, and A. Troncoso (Eds.). Springer International Publishing, Cham, 251–261.

[74] Amir Nakib and El-Ghazali Talbi (Eds.). 2017. *Metaheuristics for Medicine and Biology*. Studies in Computational Intelligence, Vol. 704. DOI: https://doi.org/10.1007/978-3-662-54428-0

[75] Ngoc Thanh Nguyen, Ryszard Kowalczyk, and Fatos Xhafa (Eds.). 2015. *Transactions on Computational Collective Intelligence XIX*. Lecture Notes in Computer Science, Vol. 9380. Springer. DOI: https://doi.org/10.1007/978-3-662-49017-4

[76] Kota Nomura and Makoto Fukumoto. 2018. Music melodies suited to multiple users' feelings composed by asynchronous distributed interactive genetic algorithm. *Int. J. Softw. Innov.* 6, 2 (2018), 26–36.

[77] K. Park, B. K. Oh, H. S. Park, and S. W. Choi. 2015. GA-based multi-objective optimization for retrofit design on a multi-core PC cluster. *Comput.-Aided Civ. Infrast. Eng.* 30, 12 (2015), 965–980. DOI: https://doi.org/10.1111/mice.12176

[78] F. Pinel, B. Dorronsoro, and P. Bouvry. 2013. Solving very large instances of the scheduling of independent tasks problem on the GPU. *J. Parallel Distrib. Comput.* 73, 1 (2013), 101–110. DOI: https://doi.org/10.1016/j.jpdc.2012.02.018

[79] Hartmut Pohlheim. 2011. GEATbx—Genetic and Evolutionary Algorithm Toolbox for use with Matlab. Retrieved from http://www.geatbx.com/.

[80] J. Porta, J. Parapar, R. Doallo, F. F. Rivera, I. Santé, and R. Crecente. 2013. High performance genetic algorithm for land use planning. *Comput. Environ. Urb. Syst.* 37 (2013), 45–58. DOI: https://doi.org/10.1016/j.compenvurbsys.2012.05.003

[81] P. Pospichal, J. Jaros, and J. Schwarz. 2010. Parallel genetic algorithm on the CUDA architecture. In *Applications of Evolutionary Computation*, C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. I. Esparcia-Alcazar, C. K. Goh, J. J. Merelo, F. Neri, M. Preuß, J. Togelius, and G. N. Yannakakis (Eds.). Springer Berlin, 442–451.

[82] T. Potuzak. 2015. Distributed/parallel genetic algorithm for road traffic network division using step parallelization. In *Proceedings of the 4th Eastern European Regional Conference on the Engineering of Computer Based Systems*. 67–74. DOI: https://doi.org/10.1109/ECBS-EERC.2015.19

[83] T. Potuzak. 2018. *Sparsely Synchronized Distributed/Parallel Genetic Algorithm for Road Traffic Network Division*. Springer International Publishing, Cham, 103–114. DOI: https://doi.org/10.1007/978-3-319-62120-3_9

[84] P. Prakash, K. G. Darshaun, M. V. Ganesh, and B. Vasudha. 2017. Fog computing: Issues, challenges and future directions. *Int. J. Elect. Comput. Eng.* 7, 6 (2017), 3669–3673.

[85] Dave Radford. 2016. *A Comparative Analysis of the Performance of Scalable Parallel Patterns Applied to Genetic Algorithms and Configured for NVIDIA GPUs*. Ph.D. Dissertation. The University of Guelph.

[86] S. Ramírez-Gallego, A. Fernández, S. García, M. Chen, and F. Herrera. 2018. Big data: Tutorial and guidelines on information and process fusion for analytics algorithms with MapReduce. *Inf. Fusion* 42 (2018), 51–61. DOI: https://doi.org/10.1016/j.inffus.2017.10.001

[87] J. Ren, Y. Pan, A. Goscinski, and R. A. Beyah. 2018. Edge computing for the internet of things. *IEEE Netw.* 32, 1 (Jan. 2018), 6–7. DOI: https://doi.org/10.1109/MNET.2018.8270624

[88] V. Roberge, M. Tarbouchi, and G. Labonte. 2013. Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning. *IEEE Trans. Industr. Inform.* 9, 1 (Feb. 2013), 132–141. DOI: https://doi.org/10.1109/TII.2012.2198665

[89] V. Roberge, M. Tarbouchi, and F. Okou. 2014. Strategies to accelerate harmonic minimization in multilevel inverters using a parallel genetic algorithm on graphical processing unit. *IEEE Trans. Power Electron.* 29, 10 (Oct. 2014), 5087–5090. DOI: https://doi.org/10.1109/TPEL.2014.2311737

[90] Ozgur Koray Sahingoz. 2014. Generation of Bezier curve-based flyable trajectories for multi-UAV systems with parallel genetic algorithm. *J. Intell. Robot. Syst.* 74, 1 (01 Apr. 2014), 499–511. DOI: https://doi.org/10.1007/s10846-013-9968-6

[91] Carolina Salto and Enrique Alba. 2015. Adapting distributed evolutionary algorithms to heterogeneous hardware. *Trans. Comput. Collect. Intell.* 19 (2015), 103–125. DOI: https://doi.org/10.1007/978-3-662-49017-4_7

[92] Carolina Salto, Gabriela Minetti, Enrique Alba, and Gabriel Luque. 2018. Developing genetic algorithms using different MapReduce frameworks: MPI vs. Hadoop. In *Advances in Artificial Intelligence*, Francisco Herrera, Sergio Damas, Rosana Montes, Sergio Alonso, Óscar Cordón, Antonio González, and Alicia Troncoso (Eds.). Springer International Publishing, Cham, 262–272.

[93] John Schroeder. 2008. Remote optimization. In *Process Automation Services & Capabilities*. ABB, 13–16.

[94] Birgitt Schönfisch and André de Roos. 1999. Synchronous and asynchronous updating in cellular automata. *Biosystems* 51, 3 (1999), 123–143. DOI: https://doi.org/10.1016/S0303-2647(99)00025-8

[95] Vitaly Shaferman and Tal Shima. 2015. Tracking multiple ground targets in urban environments using cooperating unmanned aerial vehicles. *J. Dyn. Syst. Meas. Contr.* 5, 137 (2015), 051010–051010–11. DOI: https://doi.org/10.1115/1.4028594

[96] A. Shayeghi, D. Gotz, J. B. A. Davis, R. Schafer, and R. L. Johnston. 2015. Pool-BCGA: A parallelised generation-free genetic algorithm for the ab initio global optimisation of nanoalloy clusters. *Phys. Chem. Chem. Phys.* 17, 3 (2015), 2104–2112. DOI: https://doi.org/10.1039/C4CP04323E

[97] Z. Shen, K. Wang, and F. Y. Wang. 2013. GPU based non-dominated sorting genetic algorithm-ii for multi-objective traffic light signaling optimization with agent based modeling. In *Proceedings of the 16th International IEEE Conference on Intelligent Transportation Systems (ITSC'13)*. 1840–1845. DOI: https://doi.org/10.1109/ITSC.2013.6728496

[98] Rachhpal Singh. 2017. Genetic-variable neighborhood search with thread replication for mobile cloud computing. *Int. J. Parallel, Emerg. Distrib. Syst.* 32, 5 (2017), 486–501. DOI: https://doi.org/10.1080/17445760.2016.1188386

[99] Othman Soufan, Dimitrios Kleftogiannis, Panos Kalnis, and Vladimir B. Bajic. 2015. DWFS: A wrapper feature selection tool based on a parallel genetic algorithm. *PLOS One* 10, 2 (02 2015), 1–23. DOI: https://doi.org/10.1371/journal.pone.0117988

[100] Daniel H. Stolfi and Enrique Alba. 2014. Red swarm: Reducing travel times in smart cities by using bio-inspired algorithms. *Appl. Soft Comput.* 24 (2014), 181–.195. DOI: https://doi.org/10.1016/j.asoc.2014.07.014

[101] Daniel H. Stolfi and Enrique Alba. 2017. Computing new optimized routes for GPS navigators using evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'17)*. ACM, New York, NY, 1240–1247. DOI: https://doi.org/10.1145/3071178.3071193

[102] Janko Straßburg, Christian Gonzàlez-Martel, and Vassil Alexandrov. 2012. Parallel genetic algorithms for stock market trading rules. *Proced. Comput. Sci.* 9 (2012), 1306–1313. DOI: https://doi.org/10.1016/j.procs.2012.04.143

[103] C. Sunitha and I. Jeevitha. 2015. A review on genetic algorithm practice in Hadoop MapReduce. *Int. J. Sc. Technol. Eng.* 2, 5 (12 2015), 150–155.

[104] E. Taillard. 1993. Benchmarks for basic scheduling problems. *Euro. J. Oper. Res.* 64, 2 (1993), 278–285. DOI: https://doi.org/10.1016/0377-2217(93)90182-M

[105] El-Ghazali Talbi. 2009. *Metaheuristics: From Design to Implementation*. Wiley.

[106] El-Ghazali Talbi. 2015. Parallel evolutionary combinatorial optimization. In *Handbook of Computational Intelligence*.

[107] El-Ghazali Talbi and Geir Hasle. 2013. Metaheuristics on GPUs. *J. Parallel Distrib. Comput.* 73, 1 (2013), 1–3. DOI: https://doi.org/10.1016/j.jpdc.2012.09.014

[108] D. Thomas and B. C. Kovoor. 2018. A genetic algorithm approach to autonomous smart vehicle parking system. *Proced. Comput. Sci.* 125 (2018), 68–76. DOI: https://doi.org/10.1016/j.procs.2017.12.011

[109] Marco Tomassini. 1999. Parallel and Distributed Evolutionary Algorithms: A Review. In *Evolutionary Algorithms in Engineering and Computer Science*, Kaisa Miettinen, Pekka Neittaanmäki, M. M. Mäkelä, and Jacques Périaux (Eds.). John Wiley & Sons, Inc.

[110] Umut Tosun, Tansel Dokeroglu, and Ahmet Cosar. 2013. A robust island parallel genetic algorithm for the quadratic assignment problem. *Int. J. Prod. Res.* 51, 14 (2013), 4117–4133. DOI: https://doi.org/10.1080/00207543.2012.746798

[111] Jamal Toutouh and Enrique Alba. 2017. Parallel multi-objective metaheuristics for smart communications in vehicular networks. *Soft Comput.* 21, 8 (01 Apr. 2017), 1949–1961. DOI: https://doi.org/10.1007/s00500-015-1891-2

[112] A. J. Umbarkar and M. S. Joshi. 2013. Review of parallel genetic algorithm based on computing paradigm and diversity in search space. *ICTACT J. Soft Computi.* 3, 4 (2013), 615–622. Retrieved from https://www.ingentaconnect.com/content/doaj/09766561/2013/00000003/00000004/art00007.

[113] José Valente de Oliveira, Sérgio Baltazar, and Helder Daniel. 2017. On asynchronous parallelization of order-based GA over grid-enabled heterogenous commodity hardware. *Soft Comput.* 21, 21 (1 Nov. 2017), 6351–6368. DOI: https://doi.org/10.1007/s00500-016-2190-2

[114] Abhishek Verma, Xavier Llorà, David E. Goldberg, and Roy H. Campbell. 2009. Scaling genetic algorithms using MapReduce. In *Proceedings of the 9th International Conference on Intelligent Systems Design and Applications*. IEEE, 13–18.

[115] Pablo Vidal and Enrique Alba. 2010. *Cellular Genetic Algorithm on Graphic Processing Units*. Springer Berlin, 223–232. DOI: https://doi.org/10.1007/978-3-642-12538-6_19

[116] Pablo Vidal and Enrique Alba. 2010. A multi-GPU implementation of a cellular genetic algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'10)*. 1–7. DOI: https://doi.org/10.1109/CEC.2010.5586530

[117] Kai Wang and Zhen Shen. 2012. A GPU-based parallel genetic algorithm for generating daily activity plans. *IEEE Trans. Intell. Transport. Syst.* 13, 3 (2012), 1474–1480. DOI: https://doi.org/10.1109/TITS.2012.2205147

[118] Zhenyu Wen, Renyu Yang, Peter Garraghan, Tao Lin, Jie Xu, and Michael Rovatsos. 2017. Fog orchestration for internet of things services. *IEEE Internet Comput.* 21, 2 (Mar. 2017), 16–24. DOI: https://doi.org/10.1109/MIC.2017.36

[119] Darrell Whitley and Timothy Starkweather. 1990. GENITOR II: A distributed genetic algorithm. *J. Exper. Theor. Artif. Intell.* 2, 3 (1990), 189–214. DOI: https://doi.org/10.1080/09528139008953723

[120] L. Darrell Whitley, Francisco Chicano, and Brian W. Goldman. 2016. Gray box optimization for Mk landscapes (NK landscapes and MAX-kSAT). *Evol. Comput.* 24, 3 (2016), 491–519. DOI: https://doi.org/10.1162/EVCO_a_00184

[121] Jonathan Wright and Ali Alajmi. 2005. The robustness of genetic algorithms in solving unconstrained building optimisation problems. In *Proceedings of the 7th IBPSA Conference: Building Simulation*. 15–18.

[122] Rakesh Yadav. 2015. *Genetic Algorithms Using Hadoop MapReduce*. Ph.D. Dissertation. National Institute of Technology Rourkela.

[123] C. Yang, B. Chen, and S. Zhang. 2011. Design and implementation of general integrated optimization design software SiPESC.OPT. *Computer Aided Engineering* 20 (01 2011), 42–48.

[124] C. Yang, H. Li, Y. Rezgui, I. Petri, B. Yuce, B. Chen, and B. Jayan. 2014. High throughput computing based distributed genetic algorithm for building energy consumption optimization. *Energy Build.* 76 (2014), 92–101. DOI: https://doi.org/10.1016/j.enbuild.2014.02.053

[125] Zhaosheng Yang, Duo Mei, Qingfang Yang, Huxing Zhou, and Xiaowen Li. 2014. Traffic flow prediction model for large-scale road network based on cloud computing. *Math. Prob. Eng.* 2014 (2014).

[126] Cheng-Jin Ye and Min-Xiang Huang. 2015. Multi-objective optimal power flow considering transient stability based on parallel NSGA-II. *IEEE Trans. Power Syst.* 30, 2 (Mar. 2015), 857–866. DOI: https://doi.org/10.1109/TPWRS.2014.2339352

[127] Xin-Yuan Zhang, Jun Zhang, Yue-Jiao Gong, Zhi-Hui Zhan, Wei-Neng Chen, and Yun Li. 2016. Kuhn-Munkres parallel genetic algorithm for the set cover problem and its application to large-scale wireless sensor networks. *IEEE Trans. Evol. Comput.* 20, 5 (Oct. 2016), 695–710. DOI: https://doi.org/10.1109/TEVC.2015.2511142

Received February 2019; revised March 2020; accepted May 2020