Soft Computing
https://doi.org/10.1007/s00500-019-04125-w

The grid-to-neighbourhood relationship in cellular GAs: from design to solving complex problems

Zakaria Abdelmoiz Dahi[1] · Enrique Alba[2]

Abstract
Cellular genetic algorithms (cGAs) are a class of evolutionary algorithms in which the population is structured as a grid and interactions between individuals are restricted to the neighbourhood. Like any other optimisation algorithm, the cGA's efficiency lies in its ability to find an adequate balance between its exploratory and exploitive capabilities. The search selection pressure represents a good indicator of the state of that balance. From that point of view, it has been shown that the cGA's grid-to-neighbourhood relationship can be used to reflect this property. Until today, not much has been done in that area of research and many questions still surround this grid-to-neighbourhood effect. This paper describes a systematic study on the effects of that ratio on the efficiency of the cGA. This is done by proposing a dynamic cGA that adapts its ratio through evolving its grid structure using some strategy. The study is conducted using a wide range of dynamic and static ratio-control policies and, for the first time, by considering both synchronous and asynchronous cGAs. As a validation problem, we have opted for a real-world complex problem in advanced cellular networks: the users' mobility management. A wide set of differently sized and realistic instances of this problem have been used, and several comparisons have been conducted against other top-ranked solvers. The experiments showed that the ratio strategy rules the cGA's convergence, efficiency and scalability. Its effectiveness is correlated with the ratio-adaptation policy and the replacement synchronism being used. Indeed, our proposals that are based on deterministic and dynamic strategies with an asynchronous replacement were able to outperform most of the state-of-the-art algorithms.

Keywords Cellular genetic algorithms · Adaptation · Cellular networks · Mobility management

1 Introduction

Considering the world's continuous evolution (e.g. in culture, economy, nature, etc.), humankind was always brought to perform its tasks by considering some restrictions (e.g. money, emotions, etc.) in order to fulfil a given (or many) objective(s) (e.g. earning, satisfaction, etc.). Such tasks are known as optimisation problems. Numerous efforts have been done in order to design techniques able to solve them adequately. At first, *classical* algorithms such as exact ones (e.g. branch and x family, dynamic programming, etc.) appeared. Most of them are based on an exhaustive or methodical enumeration of the set of possible solutions, which guarantees the resolving optimality. Such methods display promising efficiency when addressing low-complexity problems. But, as the problem's size gets larger, these solvers see their need for memory and time increase exponentially. These shortcomings are even more accentuated with regard to the constant increase in the complexity of optimisation problems, which comes in correlation with the evolution that all the fields are witnessing these days, where instantaneous respond and real-time solving are key factors to be considered.

These facts made classical algorithms inadequate for addressing today's challenging problems. However, in order to be able to do so, a *new* way of tackling optimisation problems appeared called *metaheuristics*. The latter are

Communicated by V. Loia.

✉ Zakaria Abdelmoiz Dahi
zakaria.dahi@univ-constantine2.dz

Enrique Alba
eat@lcc.uma.es

[1] MISC Laboratory, Dep. Fundamental Computer Sciences and their Applications, Constantine 2 University, Constantine, Algeria

[2] NEO Laboratory, Dep. de Lenguajes y Ciencias de la Computación, University of Málaga, Málaga, Spain

Published online: 21 June 2019

problem-independent algorithms that iteratively evolve a set of (or one) solutions towards fitter states by using some search operators until a given termination criterion is met. This type of solvers sacrifices the solution's optimality for the sake of a quick and suboptimal solving. Actually, unlike the classical algorithms, metaheuristics present several advantages such as having low computational complexity, low memory needs and yielding a near-optimal solution in a reasonable time even for complex real-life problems (Banharnsakun 2019; Fahad et al. 2019; Lechuga and Sánchez 2019; Torres-Escobar et al. 2019). Many classifications exist for metaheuristics, where each considers a given feature, but in general, the source of inspiration is the most commonly employed criterion. With regard to the aforementioned, metaheuristics can be categorised as Evolutionary and Swarm-based Algorithms (EAs and SAs). EAs, such as the differential evolution algorithm, are inspired from the natural selection where only the fittest individuals survive. On the other hand, SAs, like the particle swarm optimisation algorithm, take inspiration from swarm movements (Talbi 2009).

A known feature of modern search algorithms like metaheuristics is that their efficiency is highly correlated with the specialisation of their parameters tuning. In fact, in order to better adjust to the problem/instance properties, a fine-grained tuning is almost mandatory. However, such a task is costly and requires advanced knowledge of both the algorithm used and the problem being treated. This restricts the use of today's algorithms to experts in abstract research fields instead of novice users in real-life environments.

As a solution to this issue, researchers are seeking removing the human factor needed for such specialised tuning. This is generally done by designing algorithms that include some mechanisms that automatically adapt the parameters to better fit the problem/instance being solved (Pang et al. 2018). The efficiency of an adaptation strategy stands on its ability to create the best possible balance between the exploitive and exploratory capacities of the algorithm (Lin and Gen 2009). However, such a task requires two background pieces of information on the algorithm itself: the parameter driving this balance and how it does it.

The cellular genetic algorithm (cGA) is a metaheuristic that belongs to the family of structured evolutionary algorithms (sEAs) (Alba and Dorronsoro 2008) where the population is organised as a grid. Each node in the grid represents an individual, and interactions between individuals are restricted to a neighbourhood. Some studies, Alba and Dorronsoro (2005, 2008) have shown that the balance between the cGA's intensification and diversification capabilities can be affected by the grid-to-neighbourhood relationship or the so-called "*ratio*". However, despite its importance, little effort has been made to understand this relationship, its influence and to model or control it, so that to produce better working cGAs and hopefully better sEAs. Indeed, to the best of our knowledge, only one work has done it (Alba and Dorronsoro 2005). The latter is a unique and pioneer work, which made its findings extremely valuable for the research community. However, the study conducted in that work turned to have some shortcomings at the experimental level, which made the reliability and consistency of its findings incomplete. Thus, many research questions still surround this grid-to-neighbourhood relationship such as: Does the ratio really affect the cGA's search process? How it does it? Can we control it? How? Ultimately, what are the guidelines for designing an efficient ratio-driven cGA?

This paper describes a systematic study that aims to answer some of these questions by fixing the flaws in Alba and Dorronsoro (2005). This is done by proposing variants of the ratio-driven cellular genetic algorithm (RD-cGA). Unlike the classical cGA, our proposal evolves its ratio over the executions throughout adapting its grid structure according to some strategy.

In order to cope, in a consistent and reliable way, with the pitfalls in Alba and Dorronsoro (2005), we investigate the same wide range of static and dynamic policies of ratio control that have been devised in that work. In addition, we go beyond that study which has only considered the synchronous cGA, made the evaluation on the basis of few metrics and to do that, it used non-realistic low-dimensional benchmark problems with limited synthetic or randomly generated instances. Besides, that work made no further comparisons with state-of-the-art optimisers. Thus, we consider in our work both synchronous and asynchronous cGAs. For the validation, we have opted for a challenging NP-complete real-world optimisation problem in mobile networks: the mobility management of mobile users (Hać and Zhou 1997). In order to assess the efficiency, scalability and robustness of the proposed variants, several differently sized and realistic instances have been used. A comparison against state-of-the-art solvers has also been done.

The remainder of this paper is structured as follows. In Sect. 2, we introduce the basic concepts of cGA, adaptation in EAs and the "ratio" concept. Section 3 gives a profound literature review of both adaptive cGAs and EAs. In Sect. 4, we present the studied RD-cGAs. Section 5 introduces the experimental methodology. Section 6 presents and discusses the experimental results. Section 7 provides an analysis of the RD-cGAs. Finally, the paper is concluded in Sect. 8.

2 Preliminaries

In this section, we present the basic notions related to the canonical cGA, adaptation in EAs and the grid-to-neighbourhood relationship in cGAs.

2.1 Cellular genetic algorithms

The classical cGA belongs to the category of structured evolutionary algorithms, where the population is organised as a static grid with a fixed length and width of H and V nodes, respectively. Each node in the grid represents an individual that interacts with a specific number of individuals defined by a neighbourhood. Many grid shapes and neighbourhood types have been proposed in the literature. Figure 1a represents an instance of a toroidal square grid with H and V equal to 5, whereas Fig. 1b illustrates the Von Neumann (or the so-called L5 or NEWS), L9, C9, C13, C21 and C25 neighbourhood configurations (Alba and Dorronsoro 2008).

Like its classical counterpart, the cGA takes as input a set of individuals (i.e. solutions) and evolves them iteratively towards a fitter state by applying a series of operators. Unlike the basic GA, the cGA applies those operators sequentially on each individual by browsing the grid node by node. After initialisation, a classical generation of the cGA consists of applying the selection, variation operators, evaluation and replacement. The pseudocode of Algorithm 1 summarises the general framework of the basic cGA.

Algorithm 1. The Cellular Genetic Algorithm

```
1: Initialisation.
2: while (termination criterion is not reached) do
3:   for (every individual in the grid) do
4:     Selection inside neighbourhood of the individual.
5:     Variation operators: crossover and mutation.
6:     Evaluation.
7:     Replacement inside neighbourhood of the individual.
8:   end for
9: end while
```

The cGA starts by initialising a population of N individuals, where each individual $\overrightarrow{X} = \{x_1, x_2, \ldots, x_d\}$ represents a possible solution to the d-dimensional problem being addressed. Each element x_j is a problem variable to be optimised, where $j = 1, \ldots, d$. Theoretically, all initialisation strategies of the classical GA are applicable to the cGA. We can cite, for instance, the heuristic and random ones. After initialisation, the quality of all individuals is evaluated using the objective function of the problem being solved.

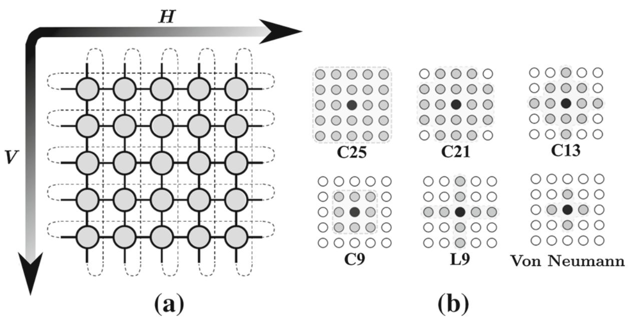

Fig. 1 **a** Grid example: 5 × 5 nodes, **b** cGAs' neighbourhood configurations: black nodes are the centres of the neighbourhoods and grey ones are the rest of the neighbourhoods' nodes

Once the initialisation phase has been achieved, the cGA selects a given number of parents M using a given strategy. Again, all selection strategies of the unstructured GA can be applied to the cGA: fitness-proportionate, tournament or ranking-based, etc.

After selection, the chosen parents go through the breeding process. The latter consists of applying the crossover and mutation operators. Both incarnate the GA's capabilities of *exploiting* the already known zones of the search domain and *exploring* other new ones. This is done by creating new individuals whose components are taken from already existing parents or created by randomness. The crossover consists of interchanging some parts between one parent and another. The segments that are concerned by this exchange and the way this swap is performed are ruled by a probability P_c and the type of crossover used. We can cite, for instance, the single-point, two-point and uniform crossovers. After applying the crossover, the mutation operator is applied to the offspring produced. It stands in altering the information that is contained in each (or some) of the offspring's genes. Like the crossover, in the mutation also, the amount of perturbation and the way it is performed are controlled by both the probability P_m and the type of mutation employed. We can mention for example the bit-flip and Cauchy mutations (Sivanandam and Deepa 2007).

Having obtained the new offspring produced by successively applying both the crossover and mutation, its quality is assessed using the objective function of the problem being addressed. Then, a replacement phase takes place to decide on the individual composition for the next iteration. It has to be noted that replacement in the cGA consists of deciding *who* and *when* the individual will be inserted in the new population. In that perspective, theoretically, all replacement strategies of the classical GA are applicable to the cGA, e.g. (λ, μ), $(\lambda + \mu)$, $(1 + 1)$ replacement strategies.

For deciding when the new individual will be inserted in the population, two main strategies exist: *synchronous* and *asynchronous*. The synchronous replacement places the newly produced individual in an auxiliary population and once all individuals in the grid have been browsed, the auxiliary population replaces the old one. On the other hand, the asynchronous replacement strategy directly replaces the individual being processed by the newly created one. It is also to be noted that there is a concrete *plan* for processing individuals in the grid.

In order to accomplish an iteration of the cGA, the whole process has to be repeated until all the nodes (individuals) of the grid are browsed. Then, the best individual B is updated.

The classical cGA iteration is repeated until a given termination criterion is reached.

2.2 Adaptation in evolutionary algorithms

Like EAs, most of today's algorithms are problem or even instance-dependent. In fact, their efficiency principally relies on the specialisation of their tuning. A fine-grained tuning of their parameters is done by practitioners in order to fit the problem/instance being solved as closely as possible. However, this task is generally performed through long and exhausting processes. Moreover, this has to be performed each time a new problem/instance is tackled. In addition, this process requires advanced knowledge of the algorithm used and the problem/instance being addressed. These facts made the use of today's algorithms costly and restricted to experts in abstract research instead of novice users in a daily task in an ordinary environment.

As a promising solution to this issue, efforts are being made to design dynamic algorithms incorporating some mechanisms that automatically adapt the values of the algorithm's parameters without any external (human) interference. Many adaptation strategies have been proposed in the literature. They can be classified as either (i) deterministic, (ii) adaptive or (iii) self-adaptive (Alba and Dorronsoro 2005).

The deterministic approach consists of adapting the value of a given parameter using a predefined deterministic formula, while in the adaptive approach the adaptation is driven by some information received from the algorithms themselves, while being run. Adaptation in the self-adaptive approach encodes the value of the parameter being adapted as part of the problem's solution and makes the whole entity evolve through iterations using the algorithm's operators. The deterministic and adaptive approaches are those studied in this paper.

2.3 Grid-to-neighbourhood relationship in cGAs

Previous studies (Alba and Dorronsoro 2005; Alba and Troya 2000; Sarma and De Jong 1996) have concluded that the cGA grid can be mathematically characterised by its radius. This is defined as the dispersion of n^* points in a circle centred in (x_c, y_c) defined by Eq. (1). A radius, R_g, of a cGA grid can be defined using Eq. (2).

$$x_c = \frac{\sum_{i=1}^{n^*} x_i}{n^*}, \quad y_c = \frac{\sum_{i=1}^{n^*} y_i}{n^*} \tag{1}$$

$$R_g = \frac{\sqrt{\sum (x_i - x_c)^2 + \sum (y_i - y_c)^2}}{n^*} \tag{2}$$

Other works by Sarma and De Jong (1996) and Alba and Dorronsoro (2005) have concluded that the radius definition can be extended to characterise the neighbourhood as well, and thus, the grid-to-neighbourhood relationship can be quantified. This has led to the appearance of the *ratio* concept. The latter can be defined using Eq. (3), where R_n and R_g represent the radius of the neighbourhood and the grid, respectively. $R_{n:g}$ corresponds to their ratio.

$$R_{n:g} = \frac{R_n}{R_g} \tag{3}$$

Furthermore, studies in Alba and Dorronsoro (2005, 2008); Sarma and Jong (1997) have shown that $R_{n:g}$ is a metric that correctly reflects the selection pressure of the cGA. In fact, as it can be seen in Fig. 2a (Alba and Dorronsoro 2005), algorithms with different neighbourhood configurations (L5 and C21) and grids (32 × 32, 64 × 64) but similar ratios,

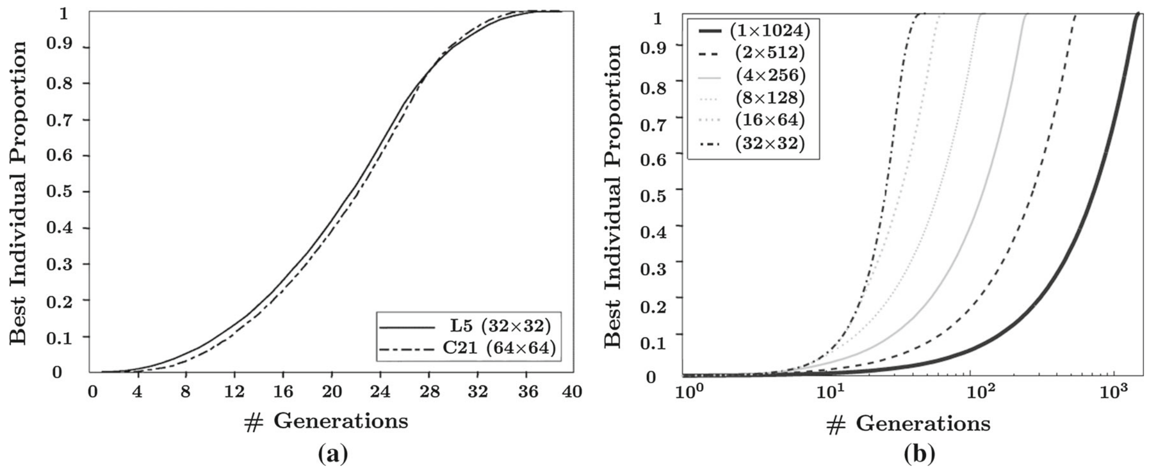

Fig. 2 Correlation between the cGAs' selection pressure evolution and the ratio value: **a** cGAs using binary tournament selection, different neighbourhoods and different grid sizes but similar ratios, **b** cGAs using binary tournament selection and similar grid sizes but different ratios

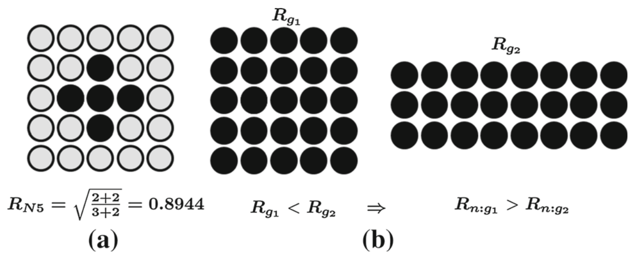
R_{g_1}
R_{g_2}
$R_{N5} = \sqrt{\frac{2+2}{3+2}} = 0.8944$
$R_{g_1} < R_{g_2} \Rightarrow R_{n:g_1} > R_{n:g_2}$
(a)
(b)

Fig. 3 Ratio and radius calculation in cGAs: neighbourhoods' nodes are in black, **a** 5×5 grid using Von Neumann neighbourhood, **b** grids having similar sizes ($5 \times 5 \simeq 3 \times 8$), but different ratios

display the same selection pressure, whereas Fig. 2b (Alba and Dorronsoro 2008) shows that algorithms with different ratios exhibit different selection pressures.

Under the light of the findings of the above-mentioned works, the authors in Alba and Dorronsoro (2005) stated that the ratio has a direct influence on the search capacities of the cGA. In fact, they found that as the grid gets narrower, the grid radius increases, while the value of $R_{\mathrm{n:g}}$ decreases and promotes the exploratory capacities of the algorithm. Figure 3a represents the radius of the Von Neumann neighbourhood, while Fig. 3b illustrates the aforementioned findings.

3 A literature review on adaptation

In this section, we perform a rigorous and systematic literature review on adaptation in both EAs and cGAs.

3.1 Adaptive evolutionary algorithms

Grefenstette's (1986) work was one of the first behind the idea of dynamically adapting the EAs' parameters. Then, further groundbreaking research on parameters' control and adaptation in EAs followed (Bäck 1993; Schraudolph and Belew 1992; Sun and Wu 1995). Since almost three decades now, numerous works, in the same line of thoughts of pioneer ones, came up.

Lately, parameters' adaptation in EAs is still being investigated and applied to several algorithms (e.g. genetic algorithm, differential evolution, evolutionary programming, etc.) and problems of different types (e.g. mono/multi-objective, dynamic, etc.) (Dahi et al. 2018; Pang et al. 2018; Tian and Gao 2019). Most of these works specialised in a given parameter, a particular EA or applied to a certain kind of problems. This makes it hard to enumerate all the existing combinations of {algorithm, parameter, problem} that have been investigated. Although, for more details, one can refer to some extensive surveys available in the literature on today's and tomorrow's trends and challenges in EAs' parameter control and self-adaptation (Eiben and Smith 2015; Karafotias et al. 2015; Zhang et al. 2012).

3.2 Adaptive cellular genetic algorithms

With regard to including or studying parameters' adaptation in cGAs, not much has been done, which left this research axis far from being understood. Actually, to the best of the authors' knowledge, barely a dozen works have done this. Besides, most of them studied specific and specialised features (e.g. in terms of cGA's parameters, cGA's and experiments' settings, problem's type, etc.) in such a way that not many points are shared between those works. This makes it difficult to link the findings of those works and therefore built consistent and helpful knowledge about adaptive cGAs.

Some works handled the cGA as an evolutionary algorithm and applied the adaptation on the basic parameters that it shares with most of the EAs. We can cite, as an example of these parameters, the mutation probability (Dahi et al. 2018), both the mutation and crossover probabilities (Jie et al. 2017) or the selection probability (Al-Naqi et al. 2010). On the other hand, other works incorporated the adaptation in cGA-specific parameters such as the neighbourhood structure (Dorronsoro and Bouvry 2011) or the elements' migration policy between neighbourhoods (Al-Naqi et al. 2011).

All the previously mentioned works have taken mono-objective problems as validation benchmarks. We can mention, for instance, the frequency modulation sound parameter identification (FMS), the system of linear equations (Al-Naqi et al. 2010), GPS attitude determination (Al-Naqi et al. 2011; Morales-Reyes et al. 2008), the mobility management of mobile users (Dahi et al. 2018), clustering problems (Jie et al. 2017), the error correcting codes, FMS, maximum cut of a graph, massively multimodal deceptive, minimum tardiness task, P-PEAKS (Dorronsoro and Bouvry 2011) and global optimisation functions like Rastrigin, Rosenbrock and Schwefel (Al-Naqi et al. 2010; Dorronsoro and Bouvry 2011). However, other works included adaptation in the cGA to face problem-dependent features in multi-objective optimisation. An adaptive queuing to solve multi-objective functions was presented in Kamkar and Akbarzadeh (2010), while the authors in Huang et al. (2015) proposed a cGA with an adaptive items' selection strategy to address the test sheet generation problem.

If few works have studied adaptation of whatsoever cGA parameters, even fewer have investigated the grid-to-neighbourhood adaptation in cGAs. Indeed, to the best of our knowledge, only two works have achieved it (Alba and Dorronsoro 2005; Morales-Reyes et al. 2008). The authors in Alba and Dorronsoro (2005) devised eight cGA variants, where three use static ratios, two variants with preprogrammed ratio adaptation and three others use dynamic ratio-adaptation strategies. They employed the same validation problems as in Dorronsoro and Bouvry (2011). Later, the authors in Morales-Reyes et al. (2008) used one of the dynamic ratio-adaptation policies devised in Alba and Dorronsoro (2005). The latter is a unique and pioneer work, which made its findings extremely valuable for the research community. However, it suffers from some shortfalls at the experimental level, which made the reliability and consistency of its discoveries incomplete.

We can mention, as an example of these pitfalls, first, only the synchronous replacement strategy has been considered, while other important and widely spread strategies exist. This allows only making specialised and restricted findings instead of general and encompassing ones. Secondly, the proposed self-cGAs were not evaluated on the basis of different metrics such as the efficiency, scalability and robustness. Indeed, only unrealistic problems have been used as validation benchmarks, which does not permit assessing properly the robustness of the self-cGAs for tackling problems of different types (e.g. random, synthetic, realistic, etc.). Moreover, the chosen validation problems are small or average-sized, which cannot permit evaluating the scalability of the self-cGAs when dealing with problems of different sizes. Again, the selected benchmark problems are relatively easy to solve since even a simple panmictic genetic algorithm could obtain "$x\%$" rate of success in the experiments conducted in Alba and Dorronsoro (2005). This does not allow assessing the efficiency of the devised self-cGAs for solving large-scale complex real-life problems. Finally, the presented self-cGAs have been compared against each other and against a basic GA, while other top-ranked solvers exist in the literature. Here again, this does not allow situating the proposed self-cGAs among state-of-the-art algorithms.

Under the light of these facts, we perform a systematic study in order to cope with the shortfalls in Alba and Dorronsoro (2005). As a matter of fact, we extend the same study by using both synchronous and asynchronous replacement strategies. We assess the proposed approaches on the basis of different aspects (e.g. efficiency, scalability and robustness). This is done by solving a real-world challenging optimisation problem in cellular networks. Also, the chosen problem instances have an increasing size (small, average and large) and different features, types and complexities (synthetic and realistic).

4 A ratio-driven cGA for the MMP

In this section, we present the proposed and studied ratio-driven cellular genetic algorithms (RD-cGAs). First, we start by introducing the mathematical formulation of the MMP. Then, we introduce the different policies used to guide the ratio of the proposed approach. Finally, we explain the general framework, as well as the steps, of the proposed RD-cGAs.

It is worth stating also, that in this work we use the term "*cellular*" with two meanings: (i) the name of the algorithms we propose and (ii) mobile communication systems. Both meanings do not relate to each other, it just happened to be so because of the proper name of the algorithms and the telecommunication concept of using cells to organise mobile networks.

4.1 Mobility management in cellular networks

The MMP in cellular networks is the process of tracking the mobile user activity across the network. This task exists in different other multi-technological wired and wireless networks (e.g. vehicular, sensor, etc.), which makes it one of the most challenging optimisation issues in today's advanced networks (Dahi 2017).

Mobility management in cellular networks relies on two elementary tasks which are location update and paging. The first task is caused by the fact that each time a user moves inside the network, a cost is involved in its mobility acknowledgement. An update action of its location is performed each time it moves, and a registration of its new location has also to be done. A second cost is generated by the network when it tries to locate a mobile user present within the network's cells. This action involves a series of send/receive transactions or the so-called paging messages. These are carried out between the network and the mobile. It is worth mentioning that generally the number of paging transactions is directly related to the number of incoming calls. Considering these facts, the practical aim of optimising the mobility task is to minimise the costs generated by both location updates and paging messages.

A mobility management scheme defines the way the location update or paging are done. Many schemes of location update exist, for instance: the never-update and always-update, although they are almost never used in real networks. Other mobility management schemes exist and are implemented in real networks (Razavi 2011), for example: the location-area and the reporting cell schemes. The latter is studied in the present work.

The reporting cell scheme labels each cell of the network as *reporting* or *non-reporting*. A mobile performs a location-update task each time it enters a reporting cell, while it can freely move in the non-reporting ones without any acknowledgement of its new location. So, using this scheme a mobile will only be paged by its last reporting cell and its neighbouring non-reporting cells. The reporting cell scheme was first proposed by Bar-Noy and Kessler (1993). It was formulated later as an optimisation problem by Hać and Zhou (1997). The authors of both works proved that the reporting cell problem (RCP) is NP-complete.

In the next sections, both the reporting cell problem and the mobility management problem refer to the same problem.

We start here a description of how are we designing a basic cGA for the MMP, to be afterwards used to study the *exploration/exploitation* balance of the RD-cGAs in a real-world problem. Let us begin first by the solution representation in the domain of the algorithm.

4.1.1 Solution representation

A possible solution of the RCP can be represented as a vector of integers described as follows. Each vector $\overrightarrow{X} = \{x_1, x_2, \ldots, x_d\}$ represents a potential configuration of the mobile network. The size d of each vector represents the number of cells in the network.

Each element x_j from $\overrightarrow{X}$ represents the state (i.e. reporting or non-reporting) of the jth cell in the network, where $j = 1 \ldots d$. If $x_j = 1$, the jth cell is labelled as a reporting one; otherwise, it is considered to be non-reporting. Figure 4 is an illustration of this representation.

Figure 5 is an example of a possible instance of the RCP solution. It represents a configuration for a network with six cells, where the 1st and 4th cells are reporting, while the other cells are non-reporting.

4.1.2 Fitness function

The RCP objective function, to be optimised, is defined in Eq. (4) (Hać and Zhou 1997).

$$\min_{\overrightarrow{X}=\{x_1,\ldots,x_d\}} \quad f(\overrightarrow{X}) = \beta * \sum_{i\in w} L_i + \sum_{j=1}^{d} P_j * V_j \tag{4}$$

The RCP fitness function f is used to assess the quality of a given solution $\overrightarrow{X}$. The value of d is the total number of cells in the network (both reporting and non-reporting), and w represents the set of only the reporting ones. The value of d is constant for all possible solutions (i.e. all network's configurations), while the one of w is specific to each solution (i.e. depends on the network's configuration).

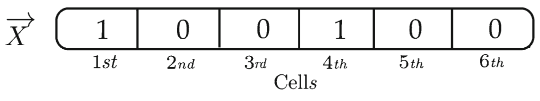

Fig. 4 A typical solution representation for the MMP

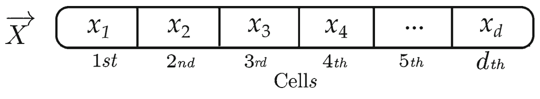

Fig. 5 Example of solution representation for the MMP: network of six cells with two reporting (1st and 4th) and four non-reporting (2nd, 3rd, 5th and 6th)

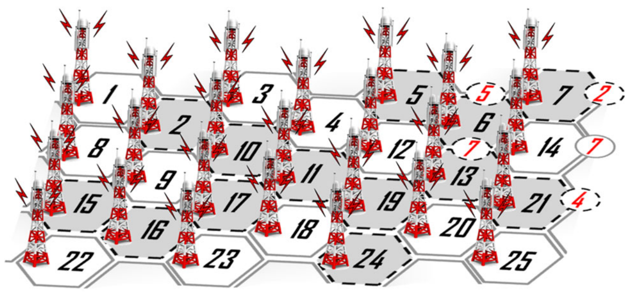

Fig. 6 Example of cell distribution: network of twenty-five cells, where thirteen are reporting (grey-shaded) and twelve non-reporting (white-shaded), numbers inside cells are the cells' ID and those inside circles are the cells' vicinity value

The variable L_i represents the number of location updates performed in the ith reporting cell, while P_j is the number of paging transactions performed within the jth cell. The value of V_j corresponds to the vicinity of the jth cell. Since the location update is considered to be 10 times more important than paging, the variable β is a weight used to balance the objective function. For reliability and comparison purposes with state-of-the-art algorithms, we set the value of β to 10 (Almeida-Luz et al. 2011; Berrocal-Plaza et al. 2014; Dahi et al. 2016; González-Álvarez et al. 2012).

It is worth noting that only the values of w and V_j are specific to each solution being evaluated and thus need to be computed each time. On the other hand, the values of β, d, L_i and P_j are given data that do not need to be calculated and are constant no matter the solution that is being assessed.

The vicinity of a cell is the maximum number of cells that have to be browsed in case an incoming call occurs in that cell. More specifically, the vicinity value of the ith reporting cell is the maximum number of non-reporting cells reachable from this cell without entering another reporting cell (including the reporting cell itself). A non-reporting cell can be reached from different reporting cells. So, we can also define the vicinity of the jth non-reporting cell as the maximum vicinity value of the reporting cells from which we can reach it.

Figure 6 represents a possible configuration of a network of 25 cells, where 13 are reporting (grey-shaded) and 12 are non-reporting cells (white-shaded). In analogy to Eq. (4), $w = 13$ and $d = 25$. Now, as an example of vicinity calculation, the vicinity of the 14th non-reporting cell is the maximum vicinity value of its neighbouring reporting cells, 6th, 7th, 13th and 21st, which are the values: 5, 2, 7 and 4, respectively. Thus, the vicinity value of the 14th non-reporting cell is 7.

4.2 The studied ratio-control policies

In this section, we introduce different policies used to control the ratio in the proposed RD-cGAs. To do so, each strategy

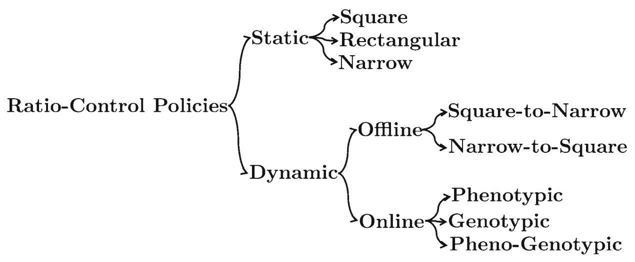

Fig. 7 Classification of the studied ratio-control strategies

follows a given criterion in order to decide *when* and *how* to adapt the RD-cGA's grid structure. We use several dynamic and static policies with different complexities and behaviour. It is to be noted that these policies are those proposed by Alba and Dorronsoro (2005) (see Fig. 7). Throughout the study, we use the Von Neumann neighbourhood and a toroidal grid.

4.2.1 Static ratio control

In this policy, the ratio of the approach is kept constant all along the run by fixing both neighbourhood and grid shape configurations. We use three static shapes of the grid: square, rectangular and narrow. Each of them is defined by a specific length H, width V and ratio $R_{\mathrm{n:g}}$. For the square grid, both H and V are equal to 20 and have an $R_{\mathrm{n:g}}$ of 0.110. For the rectangular grid, H is equal to 40 and V is equal to 10, and an $R_{\mathrm{n:g}}$ of 0.075 is used. Finally, for the narrow grid H and V are set to 100 and 4, respectively, and the ratio $R_{\mathrm{n:g}}$ is set to 0.031.

4.2.2 Dynamic ratio control

In this policy, the ratio is dynamically tuned during the run using a given strategy. In the following, we provide more details on each of these strategies.

(A) Offline adaptation

Offline adaptation uses a deterministic condition to switch the grid from a first shape A to a second shape B. The switch condition initially used in Alba and Dorronsoro (2005) consists of the half of the mean number of evaluations needed by a square-shaped cGA to solve the problem. In our work, this condition cannot be used because there is no guarantee that the square-shaped cGA can solve one or all instances. Thus, we redefine the switch condition to be half the number of iterations the approach is allowed to perform.

For the grid shapes A and B, two scenarios are considered. In the first, the grid shape changes from square-to-narrow ($R_{\mathrm{n:g}}$ shifts from 0.110 to 0.031). The second scenario is exactly the opposite.

(B) Online adaptation

This adaptation policy modifies the ratio using an adaptive approach. In this perspective, we use the three policies defined by Alba and Dorronsoro (2005): phenotypic, genotypic and the combination of both. They all have the same working principle, which is defined by Algorithm 2, where ϕ_1 and ϕ_2 are two conditions ruling the shape-shift.

Algorithm 2. Online Ratio Adaptation

```
1: if ($\phi_1$ is verified) then
2:   % ———— Promote exploitation ———— %
3:   Make the grid shape squarer.
4: else
5:   if ($\phi_2$ is verified) then
6:     % ———— Promote exploration ———— %
7:     Make the grid shape narrower.
8:   else
9:     % —— Conserve exploration/exploitation balance —— %
10:    Keep the grid shape as it is.
11:  end if
12: end if
```

Both ϕ_1 and ϕ_2 are defined according to the strategy used. In the following, we discuss each of those strategies.

- **Phenotypic Strategy**
  The conditions ϕ_1 and ϕ_2 in this strategy are defined using Eqs. (5) and (6), where $\Delta\overline{f}$ defines the difference between the mean of fitness of the individuals between iterations t and $t-1$. $\Delta\overline{f}$ is defined by Eq. (7).

$$\phi_1 ::= \Delta\overline{f}_t < (1+\epsilon).\Delta\overline{f}_{t-1} \tag{5}$$
$$\phi_2 ::= \Delta\overline{f}_t > (2-\epsilon).\Delta\overline{f}_{t-1} \tag{6}$$
$$\Delta\overline{f} = \overline{f}_t - \overline{f}_{t-1} \tag{7}$$

  If the difference $\Delta\overline{f}$ between two successive iterations t and t - 1 decreases in terms of a factor ϵ: $\Delta\overline{f}_t - \Delta\overline{f}_{t-1} < \epsilon.\Delta\overline{f}_{t-1}$, the condition ϕ_1 is fulfilled. On the other hand, if the difference $\Delta\overline{f}$ increases in terms of a factor greater than $1-\epsilon$: $\Delta\overline{f}_t - \Delta\overline{f}_{t-1} > (1-\epsilon).\Delta\overline{f}_{t-1}$, the condition ϕ_2 is verified.

- Genotypic strategy
  The conditions ϕ_1 and ϕ_2 in this strategy are defined using Eqs. (8) and (9), where $\Delta\overline{H}$ defines the difference of the average entropy of each gene in the population between iterations t and $t-1$. $\Delta\overline{H}$ is defined by Eq. (10).

$$\phi_1 ::= \Delta\overline{H}_t < (1+\epsilon).\Delta\overline{H}_{t-1} \tag{8}$$
$$\phi_2 ::= \Delta\overline{H}_t > (2-\epsilon).\Delta\overline{H}_{t-1} \tag{9}$$
$$\Delta\overline{H} = \overline{H}_t - \overline{H}_{t-1} \tag{10}$$

It is worth mentioning that this strategy is similar to the phenotypic one, but instead of evolving in terms of fitness value, the changes are ruled by the evolution in terms of Shannon entropy.

- Pheno–genotypic strategy
  Both conditions ϕ_1 and ϕ_2 in this strategy are defined as the combination of both conditions ϕ_1 and ϕ_2 presented in the phenotypic and genotypic strategies. ϕ_1 is defined using Eq. (11), while ϕ_2 is defined by Eq. (12).

$$(\Delta\overline{f}_t < (1+\epsilon).\Delta\overline{f}_{t-1}) \text{ AND } (\Delta\overline{H}_t < (1+\epsilon).\Delta\overline{H}_{t-1}) \tag{11}$$

$$(\Delta\overline{f}_t > (2-\epsilon).\Delta\overline{f}_{t-1}) \text{ AND } (\Delta\overline{H}_t > (2-\epsilon).\Delta\overline{H}_{t-1}) \tag{12}$$

4.3 The search process

In this paper, we propose and study eight variants of the ratio-driven cellular genetic algorithm (RD-cGA), where each variant is based on one of the aforementioned policies of ratio control. All the studied variants share the same framework. This is summarised by the pseudocode of Algorithm 3.

Algorithm 3. The RD-cGA Pseudocode

```
1: while (stop criterion is not reached) do
2:   for $i$=1:$V$ do
3:     for $j$=1:$H$ do
4:       % ———— Form the $(i,j)^{th}$ couple ———— %
5:       Select individual at position $(i,j)$ as $1^{st}$ parent.
6:       Select the $2^{nd}$ parent by performing a binary tournament selection on the Von Neumann neighbourhood of the $1^{st}$ parent.
7:       % ———— Crossover ———— %
8:       Apply DPX1 crossover on the $(i,j)^{th}$ couple.
9:       % ———— Mutation ———— %
10:      Apply bit-flip mutation on produced offspring.
11:      % ———— Evaluation ———— %
12:      Compute the fitness value of the produced offspring.
13:      % ———— Replacement: Asynchronous ———— %
14:      Compare the parent at position $(i,j)$ and the newly produced offspring, keep the latter if its fitness is better or equal than the parent's one.
15:      %———— Replacement: Synchronous (a) ————%
16:      Compare the parent at position $(i,j)$ and the newly produced offspring and copy the latter to an auxiliary population if its fitness is better or equal than the parent's one.
17:    end for
18:  end for
19:  %———— Replacement: Synchronous (b) ————%
20:  Consider the auxiliary population as the new population.
21:  %———— Ratio Adaptation ————%
22:  if (adaptation condition is fulfilled) then
23:    Adapt the ratio value using a ratio-control policy.
24:  end if
25: end while
```

After initialisation, a classical iteration of the RD-cGA variants consists of sequentially applying on each individual in the grid: the selection, crossover, mutation, evaluation and the replacement operators. Below, we discuss each of these phases.

4.3.1 Initialisation

A first population of N binary individuals is randomly generated. Then, its quality is evaluated using the objective function of the problem being tackled and the best individual B is extracted. Since the cGA's population is organised as a grid of H length and V width, the rest of the cGA phases are applied sequentially on each individual of that grid starting from the first one.

4.3.2 Selection

For the (i, j)th individual being processed, where $i = 1 \ldots V$ and $j = 1 \ldots H$, a couple is formed by two parents, where the 1st parent is the individual being browsed and the 2nd parent is obtained by performing a binary tournament selection on the Von Neumann neighbourhood of the (i, j)th individual.

4.3.3 Crossover

Once the (i, j)th couple has been created, the first variation operator is applied, the crossover. A random number ω is generated from a standard uniform distribution. Then, if $\omega \leqslant P_c$, the DPX1 crossover is applied, otherwise the parents of the (i, j)th couple are considered as the new offspring.

Technically speaking, the DPX1 crossover is a variant of the two-point crossover that produces only one offspring. In fact, a first step of the DPX1 consists of applying a two-point crossover. The latter stands in generating two switching points ψ_1 and ψ_2 from the interval $[1,d]$. After that, the parts of parents of the (i, j)th couple are swapped at these points. In a second phase, the offspring produced are compared to the best parent of the (i, j)th couple and the closest one in terms of Hamming distance is kept.

Figure 8 illustrates an example of how the DPX1 crossover is applied on parents of five genes each. As it can be seen, the second offspring is the one selected since it has two bits of difference with the best parent, while the first offspring has three.

4.3.4 Mutation

Mutation is performed on the offspring produced by crossover. First, for each gene of the produced offspring, a random number ω is generated from a standard uniform distribution.

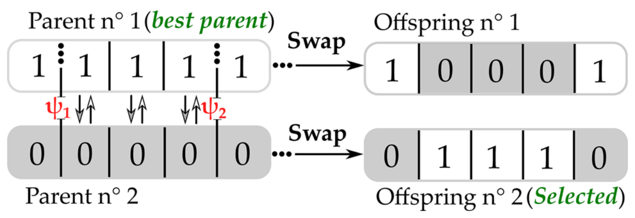

Fig. 8 DPX1 crossover: parents of five genes, 1st parent (white-shaded) and 2nd parent (grey-shaded) are swapped at (ψ_1, ψ_2) and 2nd offspring is the one selected by DPX1

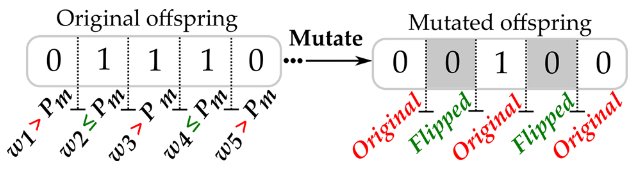

Fig. 9 Bit-flip mutation: offspring of five genes, original genes are white-shaded and flipped ones are grey-shaded

Then, if $\omega \leqslant P_\mathrm{m}$, a bit-flip mutation is applied on that gene, otherwise nothing is done.

Figure 9 illustrates how the bit-flip mutation is applied to an offspring of five bits, where two have been flipped, while the remaining ones are kept as they are.

4.3.5 Evaluation

At this step, the quality of the offspring produced after applying both crossover and mutation is assessed. This is done by computing the fitness value using the MMP's objective function defined by Eq. (4).

4.3.6 Replacement

In this phase, the composition of the population for the next iteration is decided. For the case of the RD-cGA algorithm, the replacement phase can be done according to two principal schemes: synchronous and asynchronous. In the first scheme, the offspring produced and the (i, j)th individual being processed are compared and the best one is copied to an auxiliary population. Once all the individuals in the population have been browsed, the auxiliary population is considered to be the new one. In the second approach, asynchronous, both the offspring and the (i, j)th individual being processed are compared and the best one is directly considered in the current population.

Once the replacement phase has been achieved, the whole process is repeated on the next individual in the grid until all individuals are processed.

4.3.7 Ratio adaptation

After processing all the grid's nodes, a final step consists of adapting (or not) the ratio value using one of the strategies explained in Sect. 4.2. Each strategy evolves (or not) the RD-cGA's grid structure according to some criterion. The termination of this phase marks the end of an RD-cGA iteration. Then, a whole new one is performed by applying all the aforementioned phases again.

The use of each of the above-introduced policies for ratio control gave birth to eight RD-cGA variants. The first three variants called S-cGA, R-cGA and N-cGA are based on the static square, rectangular and narrow grid shapes, respectively. The two other variants, named SN-cGA and NS-cGA, are based on the two dynamic offline strategies: square-to-narrow and narrow-to-square. The three remaining variants are called P-cGA, G-cGA and PG-cGA, which are based on the dynamic online phenotypic, genotypic and pheno–genotypic strategies, respectively.

5 Experimental design and methodology

Our experiments have been run on a cluster of nineteen heterogeneous machines with a total of 94 cores, sixteen machines with 3 cores, one machine with 8 cores and two machines with 48 cores each. The cluster is managed by the framework HTCondor.[1] All machines are running under a Linux OS. Implementation has been done using MATLAB R2014a.

The scalability, efficiency and robustness of the proposed RD-cGA variants have been assessed in twelve realistic instances with different sizes. Three networks of each size: 4×4, 6×6, 8×8 and 10×10 cells. They have been previously used in Almeida-Luz et al. (2011), Berrocal-Plaza et al. (2014), Dahi et al. (2016), González-Álvarez et al. (2012) and are available at (MMP 2008).

For consistency and comparison purposes with state-of-the-art solvers, we use the same experimental parameters as those works. Thus, the experiments have been performed until reaching the termination criterion of 175,000 fitness evaluations. We have used a population of 400 individuals, $\simeq 437$ iterations and experiments have been repeated over 30 independent runs. Several results are reported such as the best and the worst fitness values, and the mean and deviation of fitness.

On the basis of the work conducted by Alba and Dorronsoro (2005), for all the RD-cGA variants, the crossover probability P_c is set to 1, while the mutation probability P_m is set to $(1/d)$. As explained in Sects. 2.1 and 4.3 , the probabilities P_c and P_m control how many times the RD-cGA's

[1] High Throughput Computing (HTC).

crossover and mutation are performed. Now, for the P-cGA, G-cGA and PG-cGA variants, the parameter ϵ is set to 0.05.

6 Numerical results and analysis

In this section, we present the numerical results obtained from using the RD-cGA variants for addressing Networks 1, 2 and 3 with 4×4, 6×6, 8×8 and 10×10 sized cells. First, we start by presenting the results obtained by synchronous RD-cGAs. Then, we present the asynchronous ones. Finally, we compare the best synchronous and asynchronous RD-cGA variants against some of the top-ranked state-of-the-art algorithms.

For a further comparison, the RD-cGA variants are first ranked on the basis of the best solution achieved over 30 executions (Tables 1 and 4). Then, they are ranked according to the mean of fitness values (Tables 2, 3, 5 and 6). On the basis of the mean of those results, the best algorithms are highlighted in bold.

It is also to be noted that in the following sections, the metric "*# Optima*" refers to the number of the best solutions (*known in the literature*) reached by a given algorithm, while the symbol "–" is used whenever the corresponding value is unavailable in the literature. For comparison purposes with state-of-the-art solvers, the deviation in Tables 2, 3, 5 and 6 is expressed in terms of percentage. It is computed using Eq. (13).

$$\mathrm{Dev}(\%) = \left(\frac{\mathrm{Mean}}{\mathrm{Best}} - 1\right) * 100 \tag{13}$$

For readability purposes, the radius value displayed in Fig. 18 is recorded each 12 iterations $\simeq$ 4800 fitness evaluations.

Table 1 Synchronous RD-cGAs: number of best solutions reached for networks with sizes 4×4, 6×6, 8×8 and 10×10 cells

Size	Network	S-cGA	R-cGA	N-cGA	SN-cGA	NS-cGA	P-cGA	G-cGA	PG-cGA
4×4	1	**98535**	**98535**	**98535**	**98535**	**98535**	**98535**	**98535**	**98535**
	2	**97156**	**97156**	**97156**	**97156**	**97156**	**97156**	**97156**	**97156**
	3	**95038**	**95038**	**95038**	**95038**	**95038**	**95038**	**95038**	**95038**
6×6	1	**173701**	**173701**	**173701**	**173701**	**173701**	**173701**	**173701**	**173701**
	2	**182331**	**182331**	**182331**	**182331**	**182331**	**182331**	**182331**	**182331**
	3	**174519**	**174519**	**174519**	**174519**	**174519**	**174519**	**174519**	**174519**
8×8	1	**307695**	308401	308401	**307695**	308401	308401	**307695**	**307695**
	2	**287149**	**287149**	**287149**	**287149**	**287149**	**287149**	**287149**	**287149**
	3	**264204**	**264204**	**264204**	**264204**	**264204**	**264204**	**264204**	**264204**
10×10	1	387788	387060	386418	386474	387060	387206	386474	386474
	2	358414	358167	358431	358503	358503	358167	358431	358167
	3	371757	372485	371331	371584	371164	371540	**370868**	371134
# **Optima**		9	8	8	9	8	8	**10**	9

Best results are highlighted in bold

Table 2 Synchronous RD-cGAs: results obtained when tackling Networks 1, 2 and 3 with sizes 4×4 and 6×6 cells

	4×4 cells					6×6 cells				
Network	Algorithm	Best	Worst	Mean	Dev (%)	Algorithm	Best	Worst	Mean	Dev (%)
1	S-cGA	98535	98535	**98535.00**	0.00	S-cGA	173701	175241	173803.67	0.06
	R-cGA	98535	98535	**98535.00**	0.00	R-cGA	173701	175241	173758.04	0.03
	N-cGA	98535	98535	**98535.00**	0.00	N-cGA	173701	175241	173752.33	0.03
	SN-cGA	98535	98535	**98535.00**	0.00	SN-cGA	173701	175241	173803.67	0.06
	NS-cGA	98535	98535	**98535.00**	0.00	NS-cGA	173701	175241	173756.00	0.03
	P-cGA	98535	98535	**98535.00**	0.00	P-cGA	173701	175241	173807.21	0.06
	G-cGA	98535	98535	**98535.00**	0.00	G-cGA	173701	173701	**173701.00**	0.00
	PG-cGA	98535	98535	**98535.00**	0.00	PG-cGA	173701	175241	173754.10	0.03
2	S-cGA	97156	97156	**97156.00**	0.00	S-cGA	182331	182331	**182331.00**	0.00
	R-cGA	97156	97156	**97156.00**	0.00	R-cGA	182331	182331	**182331.00**	0.00
	N-cGA	97156	97156	**97156.00**	0.00	N-cGA	182331	182331	**182331.00**	0.00
	SN-cGA	97156	97156	**97156.00**	0.00	SN-cGA	182331	182331	**182331.00**	0.00
	NS-cGA	97156	97156	**97156.00**	0.00	NS-cGA	182331	182331	**182331.00**	0.00
	P-cGA	97156	97156	**97156.00**	0.00	P-cGA	182331	182331	**182331.00**	0.00
	G-cGA	97156	97156	**97156.00**	0.00	G-cGA	182331	182331	**182331.00**	0.00
	PG-cGA	97156	97156	**97156.00**	0.00	PG-cGA	182331	182331	**182331.00**	0.00
3	S-cGA	95038	95038	**95038.00**	0.00	S-cGA	174519	174519	**174519.00**	0.00
	R-cGA	95038	95038	**95038.00**	0.00	R-cGA	174519	174519	**174519.00**	0.00
	N-cGA	95038	95038	**95038.00**	0.00	N-cGA	174519	174519	**174519.00**	0.00
	SN-cGA	95038	95038	**95038.00**	0.00	SN-cGA	174519	174519	**174519.00**	0.00
	NS-cGA	95038	95038	**95038.00**	0.00	NS-cGA	174519	174519	**174519.00**	0.00
	P-cGA	95038	95038	**95038.00**	0.00	P-cGA	174519	177203	174611.55	0.05
	G-cGA	95038	95038	**95038.00**	0.00	G-cGA	174519	176847	174596.60	0.05
	PG-cGA	95038	95038	**95038.00**	0.00	PG-cGA	174519	174519	**174519.00**	0.00

Best results are highlighted in bold

Table 3 Synchronous RD-cGAs: results obtained when tackling Networks 1, 2 and 3 with sizes 8 × 8 and 10 × 10 cells

	8×8 cells					10×10 cells				
Network	**Algorithm**	**Best**	**Worst**	**Mean**	**Dev (%)**	**Algorithm**	**Best**	**Worst**	**Mean**	**Dev (%)**
1	S-cGA	307695	313700	311145.37	1.12	S-cGA	387788	399042	390729.03	0.76
	R-cGA	308401	312355	310954.00	0.83	R-cGA	387060	394510	390253.67	0.83
	N-cGA	308401	312607	**310478.43**	0.67	N-cGA	386418	396819	389475.10	0.79
	SN-cGA	307695	312906	311268.00	1.16	SN-cGA	386474	402832	390991.70	1.17
	NS-cGA	308401	312869	311104.46	0.88	NS-cGA	387060	392553	**389366.18**	0.60
	P-cGA	308401	314046	311235.24	0.92	P-cGA	387206	398410	389468.72	0.58
	G-cGA	307695	313585	311017.03	1.08	G-cGA	386474	400791	390363.13	1.01
	PG-cGA	307695	313830	311152.86	1.12	PG-cGA	386474	397332	390267.38	0.98
2	S-cGA	287149	298215	289831.57	0.93	S-cGA	358414	368620	361969.80	0.99
	R-cGA	287149	296580	290283.37	1.09	R-cGA	358167	365540	362095.56	1.10
	N-cGA	287149	297267	289976.70	0.99	N-cGA	358431	365483	361903.73	0.97
	SN-cGA	287149	299211	290985.73	1.34	SN-cGA	358503	367336	362035.30	0.99
	NS-cGA	287149	296191	**288757.07**	0.56	NS-cGA	358503	366797	361333.39	0.79
	P-cGA	287149	299564	290289.14	1.09	P-cGA	358167	364696	**361241.00**	0.86
	G-cGA	287149	301393	290628.97	1.21	G-cGA	358431	367830	362417.97	1.11
	PG-cGA	287149	298200	289913.35	0.96	PG-cGA	358167	367639	361449.00	0.92
3	S-cGA	264204	267041	264990.13	0.30	S-cGA	371757	381831	376325.63	1.23
	R-cGA	264204	267781	264946.63	0.28	R-cGA	372485	382172	376278.33	1.02
	N-cGA	264204	266517	**264512.93**	0.12	N-cGA	371331	379775	375786.13	1.20
	SN-cGA	264204	271069	265107.77	0.34	SN-cGA	371584	380990	376841.80	1.42
	NS-cGA	264204	269017	264859.93	0.25	NS-cGA	371164	381550	375980.18	1.30
	P-cGA	264204	267606	264747.14	0.21	P-cGA	371540	382638	376609.48	1.37
	G-cGA	264204	267041	265077.27	0.33	G-cGA	370868	381508	**375757.23**	1.32
	PG-cGA	264204	267781	264681.76	0.18	PG-cGA	371134	381066	376016.86	1.32

Best results are highlighted in bold

Table 4 Asynchronous RD-cGAs: number of best solutions reached for networks with sizes of 4 × 4, 6 × 6, 8 × 8 and 10 × 10 cells

Size	**Network**	**S-cGA**	**R-cGA**	**N-cGA**	**SN-cGA**	**NS-cGA**	**P-cGA**	**G-cGA**	**PG-cGA**
4×4	**1**	**98535**	**98535**	**98535**	**98535**	**98535**	**98535**	**98535**	**98535**
	2	**97156**	**97156**	**97156**	**97156**	**97156**	**97156**	**97156**	**97156**
	3	**95038**	**95038**	**95038**	**95038**	**95038**	**95038**	**95038**	**95038**
6×6	**1**	**173701**	**173701**	**173701**	**173701**	**173701**	**173701**	**173701**	**173701**
	2	**182331**	**182331**	**182331**	**182331**	**182331**	**182331**	**182331**	**182331**
	3	**174519**	**174519**	**174519**	**174519**	**174519**	**174519**	**174519**	**174519**
8×8	**1**	308401	308401	**307695**	308401	**307695**	308401	308401	**307695**
	2	**287149**	**287149**	**287149**	**287149**	**287149**	**287149**	**287149**	**287149**
	3	**264204**	**264204**	**264204**	**264204**	**264204**	**264204**	**264204**	**264204**
10×10	**1**	386731	387094	386351	386474	387462	386670	386351	386474
	2	358431	358167	358167	359330	358431	358167	358583	359577
	3	371203	371134	371164	371630	371203	371203	370966	370966
# Optima		8	8	**9**	8	**9**	8	8	**9**

Best results are highlighted in bold

6.1 Dynamic versus static synchronous RD-cGAs

In this section, we investigate how the ratio-adaptation strategy influences on the synchronous cGAs efficiency and ultimately deduce which strategy is the best.

As it can be seen from Table 1, the best RD-cGA variant is the G-cGA. When considering the "Mean" metric in Tables 2 and 3 , all variants achieve the same results for networks of size 4 × 4 cells and Network 2 of size 6 × 6 cells. Taking Network 1 of size 6 × 6 cells, the G-cGA is assessed to be the best RD-cGA, while for Network 3, all the variants have achieved the same results except the G-cGA and P-cGA. For larger networks with sizes of 8 × 8 and 10 × 10 cells, the NS-cGA has been assessed to be the best variant when tackling Networks 1 of sizes 8 × 8 and 10 × 10 cells, whereas for Network 3 with 8 × 8 cells, the N-cGA is the best variant. Finally, the P-cGA and G-cGA have been the best variants when tackling both Networks 2 and 3 with a size of 10 × 10 cells.

Considering the "Dev (%)" metric in Tables 2 and 3, all variants display approximately the same deviation when tackling Networks 1, 2 and 3 of sizes 4 × 4 and 6 × 6 cells. However, as the size of the instance increases, difference in the deviation becomes more noticeable. In fact, for all networks with sizes 8 × 8 cells and 10 × 10 cells, the SN-cGA, P-cGA and G-cGA have given the largest deviation. This tends to indicate that those variants are less stable when tackling higher instances.

Another thing to note is that when tackling low-sized instances (4 × 4 and 6 × 6 cells), all ratio-control policies achieve similar results. However, as the size of the instances increases (8×8 and 10×10 cells), the RD-cGA variants based on dynamic online ratio-control strategies become less efficient than the variants based on the remaining ratio-control

Table 5 Asynchronous RD-cGAs: results obtained when tackling Networks 1, 2 and 3 with sizes 4 × 4 and 6 × 6 cells

	4×4 cells					6×6 cells				
Network	Algorithm	Best	Worst	Mean	Dev (%)	Algorithm	Best	Worst	Mean	Dev (%)
1	S-cGA	98535	98535	**98535.00**	0.00	S-cGA	173701	175241	173906.33	0.12
	R-cGA	98535	98535	**98535.00**	0.00	R-cGA	173701	175241	173752.33	0.03
	N-cGA	98535	98535	**98535.00**	0.00	N-cGA	173701	173701	**173701.00**	0.00
	SN-cGA	98535	98535	**98535.00**	0.00	SN-cGA	173701	175241	173906.33	0.12
	NS-cGA	98535	98535	**98535.00**	0.00	NS-cGA	173701	175241	173752.33	0.03
	P-cGA	98535	98535	**98535.00**	0.00	P-cGA	173701	175241	173752.33	0.03
	G-cGA	98535	98535	**98535.00**	0.00	G-cGA	173701	177889	174045.93	0.20
	PG-cGA	98535	98535	**98535.00**	0.00	PG-cGA	173701	175241	174009.00	0.18
2	S-cGA	97156	97156	**97156.00**	0.00	S-cGA	182331	182331	**182331.00**	0.00
	R-cGA	97156	97156	**97156.00**	0.00	R-cGA	182331	182331	**182331.00**	0.00
	N-cGA	97156	97156	**97156.00**	0.00	N-cGA	182331	182331	**182331.00**	0.00
	SN-cGA	97156	97156	**97156.00**	0.00	SN-cGA	182331	182331	**182331.00**	0.00
	NS-cGA	97156	97156	**97156.00**	0.00	NS-cGA	182331	182331	**182331.00**	0.00
	P-cGA	97156	97156	**97156.00**	0.00	P-cGA	182331	182331	**182331.00**	0.00
	G-cGA	97156	97156	**97156.00**	0.00	G-cGA	182331	182331	**182331.00**	0.00
	PG-cGA	97156	97156	**97156.00**	0.00	PG-cGA	182331	182331	**182331.00**	0.00
3	S-cGA	95038	95038	**95038.00**	0.00	S-cGA	174519	178200	174641.70	0.07
	R-cGA	95038	95038	**95038.00**	0.00	R-cGA	174519	175504	174551.83	0.02
	N-cGA	95038	95038	**95038.00**	0.00	N-cGA	174519	174519	**174519.00**	0.00
	SN-cGA	95038	95038	**95038.00**	0.00	SN-cGA	174519	174519	**174519.00**	0.00
	NS-cGA	95038	95038	**95038.00**	0.00	NS-cGA	174519	175504	174551.83	0.02
	P-cGA	95038	95038	**95038.00**	0.00	P-cGA	174519	175504	174551.83	0.02
	G-cGA	95038	95038	**95038.00**	0.00	G-cGA	174519	175504	174551.83	0.02
	PG-cGA	95038	95038	**95038.00**	0.00	PG-cGA	174519	174519	**174519.00**	0.00

Best results are highlighted in bold

Table 6 Asynchronous RD-cGAs: results obtained when tackling Networks 1, 2 and 3 with sizes 8 × 8 and 10 × 10 cells

	8×8 cells					10×10 cells				
Network	Algorithm	Best	Worst	Mean	Dev (%)	Algorithm	Best	Worst	Mean	Dev (%)
1	S-cGA	308401	313571	311574.00	1.03	S-cGA	386731	404044	390405.03	0.95
	R-cGA	308401	313787	311623.80	1.05	R-cGA	387094	398654	390597.43	0.91
	N-cGA	307695	313312	**310675.50**	0.97	N-cGA	386351	393267	**389233.10**	0.75
	SN-cGA	308401	313721	311502.33	1.01	SN-cGA	386474	400047	391746.67	1.36
	NS-cGA	307695	313905	310992.93	1.07	NS-cGA	387462	397571	390379.97	0.75
	P-cGA	308401	313845	311296.73	0.94	P-cGA	386670	393922	390560.57	1.01
	G-cGA	308401	313686	311109.20	0.88	G-cGA	386351	396468	390493.60	1.07
	PG-cGA	307695	312871	311180.07	1.13	PG-cGA	386474	395030	389705.93	0.84
2	S-cGA	287149	298893	291501.47	1.52	S-cGA	358431	370437	362928.17	1.25
	R-cGA	287149	298645	290051.47	1.01	R-cGA	358167	367548	362531.07	1.22
	N-cGA	287149	297756	**289105.50**	0.68	N-cGA	358167	363339	**360722.33**	0.71
	SN-cGA	287149	301456	291302.47	1.45	SN-cGA	359330	365643	362123.93	0.78
	NS-cGA	287149	298560	289704.47	0.89	NS-cGA	358431	366880	362233.33	1.06
	P-cGA	287149	296580	289131.57	0.69	P-cGA	358167	365387	361242.80	0.86
	G-cGA	287149	299763	290595.57	1.20	G-cGA	358583	366390	361832.27	0.91
	PG-cGA	287149	296963	289910.13	0.96	PG-cGA	359577	364194	361882.20	0.64
3	S-cGA	264204	271336	265573.50	0.52	S-cGA	371203	382101	377904.07	1.81
	R-cGA	264204	270016	265119.50	0.35	R-cGA	371134	381645	376813.30	1.53
	N-cGA	264204	266624	264903.10	0.26	N-cGA	371164	379987	**375346.17**	1.13
	SN-cGA	264204	268426	265270.63	0.40	SN-cGA	371630	382158	377269.90	1.52
	NS-cGA	264204	266517	264607.40	0.15	NS-cGA	371203	380796	375348.03	1.12
	P-cGA	264204	270045	265355.77	0.44	P-cGA	371203	379711	376857.23	1.52
	G-cGA	264204	269017	265182.87	0.37	G-cGA	370966	382749	376515.03	1.50
	PG-cGA	264204	265324	**264540.50**	0.13	PG-cGA	370966	381577	375443.67	1.21

Best results are highlighted in bold

policies. This can partially be explained by the fact that as the size of the instances increases, the stability of P-cGA and G-cGA starts to decrease. It is obvious that the use of these control policies has led to the loss of balance between the exploratory and exploitive capacities of these variants, while the use of the static ratio or dynamic offline policies seems to conserve the dynamics of the RD-cGAs.

Figures 10, 11, 12 and 13 illustrate the fitness value evolution when using the synchronous RD-cGA variants for solving Networks 1, 2 and 3 with sizes 4 × 4, 6 × 6, 8 × 8 and 10 × 10 cells. It is to be noted that the fitness value evolution displayed in Figs. 10, 11, 12 and 13 is the best execution performed by the RD-cGA variants. For networks of sizes 4 × 4, 6 × 6 and 8 × 8 cells, all RD-cGAs reach the same fitness value at the end of the execution. The aim is to analyse the behaviour of several variants that obtain the same result. Moreover, these figures have been chosen to be representative (but not identical to) of the rest of instances.

On the basis of the fitness value shown in Figs. 10, 11, 12 and 13, one can see that for small networks (sizes 4 × 4 and 6 × 6 cells), all RD-cGA variants have the same convergence rate and shown the same behaviour. However, as the size

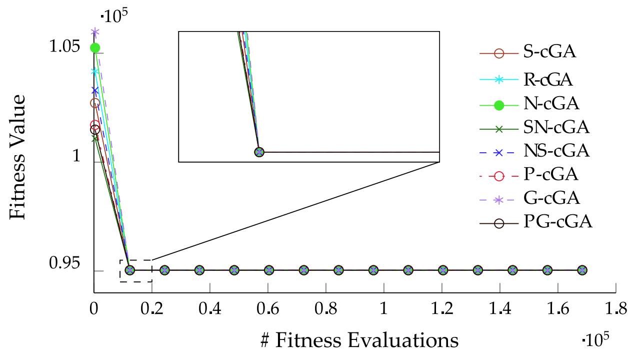

Fig. 10 Synchronous RD-cGAs' fitness value evolution: network 3 of size 4 × 4 cells

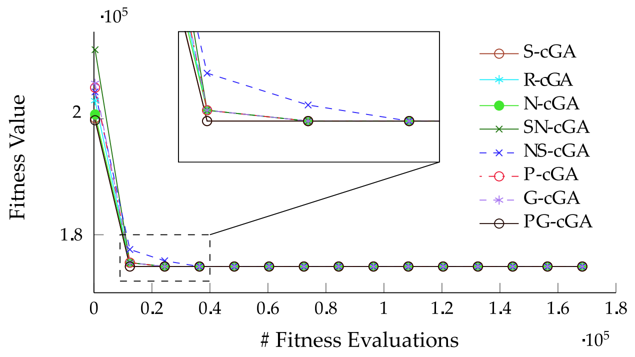

Fig. 11 Synchronous RD-cGAs' fitness value evolution: network 3 of size 6 × 6 cells

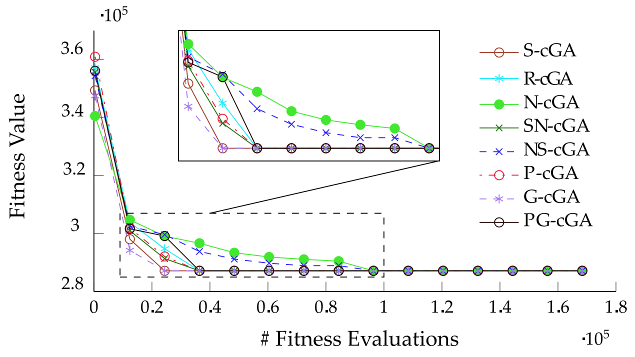

Fig. 12 Synchronous RD-cGAs' fitness value evolution: network 2 of size 8 × 8 cells

of the network increases, the difference in the convergence rate becomes more noticeable. In fact, even if all the variants reach the same result, their convergence rates differ.

6.2 Dynamic versus static asynchronous RD-cGAs

In this section, we seek to find how the ratio-adaptation strategy influences the asynchronous cGAs efficiency, and ultimately deduce which strategy is the best.

According to the results shown in Table 4, the best asynchronous RD-cGA variants are the N-cGA, NS-cGA and the PG-cGA. When considering the "Mean" metric in Tables 5 and 6 , all the RD-cGAs have achieved the same results when tackling Networks 1, 2 and 3 with size 4 × 4 cells and Network 2 with size 6 × 6 cells. However, when tackling Network 1 with size 6 × 6 cells, Networks 1 and 2 with size 8 × 8 cells and all networks with size 10 × 10 cells, the N-cGA has been the best variant. For Network 3 with size 8 × 8, the best RD-cGA is the PG-cGA. Finally, when considering Network 3 with size 6 × 6 cells, it can be seen that the best variants are the N-cGA, SN-cGA and the PG-cGA.

With respect to the metric "Dev(%)" in Tables 5 and 6, all variants have given the same deviation when tackling networks with sizes 4 × 4 and 6 × 6 cells. However, when the size of the network increases (8 × 8 and 10 × 10 cells), the difference in the deviation values between the RD-cGAs becomes more noticeable. Actually, for all networks with sizes 8 × 8 and 10 × 10 cells, it can be noted that all the RD-cGAs based on an online or offline adaptation strategy have given higher deviation than the best variant N-cGA. This partially explains the decrease of efficiency of the dynamic RD-cGAs. In fact, one can conclude that the stability of these variants is weak when tackling higher instances, which make their scalability and efficiency decline.

Like the case for the synchronous RD-cGA variants, the asynchronous RD-cGAs also achieve the same results when tackling low-dimension networks (4 × 4 and 6 × 6 cells). However, as the size of the networks increases (8 × 8 and 10 × 10 cells), the scalability of some RD-cGA variants starts to decrease. Actually, all variants based on a dynamic online strategy see their efficiency decrease when compared to RD-cGAs based on static or dynamic offline ratio-control strategies. One can explain this regression by the fact that these RD-cGAs witness a premature or slow convergence, which is principally due to an unbalance between their exploratory and exploitive capabilities.

Figures 14, 15, 16 and 17 illustrate the fitness value evolution when using the asynchronous RD-cGA variants for tackling Networks 1, 2 and 3 with sizes 4 × 4, 6 × 6, 8 × 8

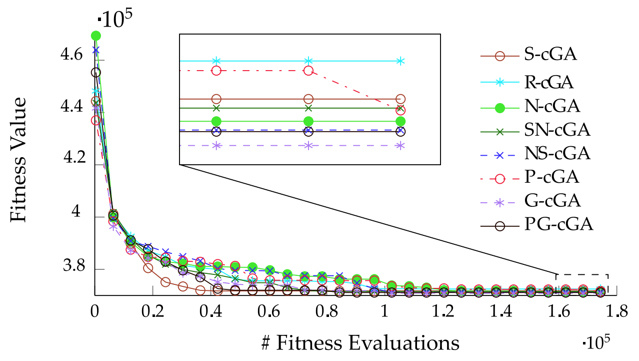

Fig. 13 Synchronous RD-cGAs' fitness value evolution: network 3 of size 10 × 10 cells

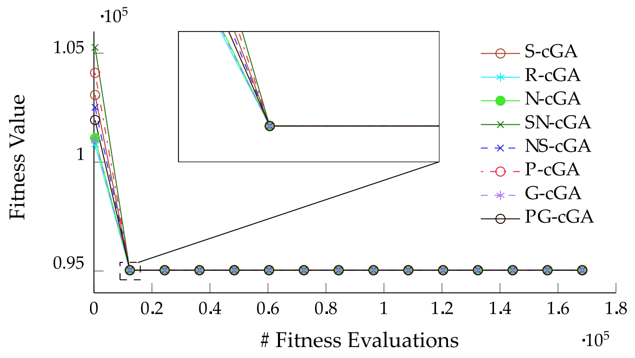

Fig. 14 Asynchronous RD-cGAs' fitness value evolution: network 3 of size 4 × 4 cells

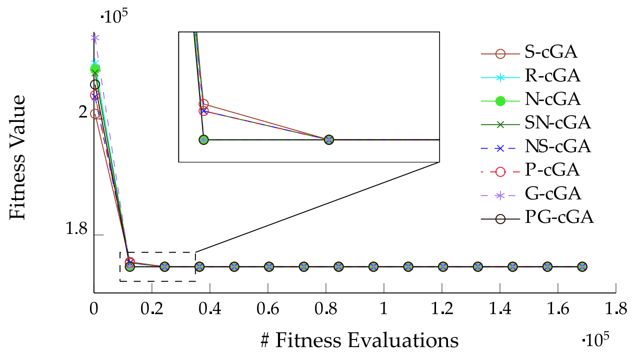

Fig. 15 Asynchronous RD-cGAs' fitness value evolution: network 3 of size 6 × 6 cells

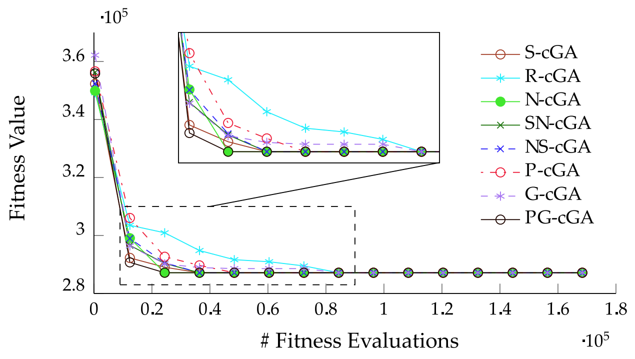

Fig. 16 Asynchronous RD-cGAs' fitness value evolution: network 2 of size 8 × 8 cells

and 10 × 10 cells. It is to be noted that the fitness value evolution displayed in Figs. 14, 15, 16 and 17 is the one of the best executions performed by the RD-cGA variants. Also, for networks of sizes 4 × 4, 6 × 6 and 8 × 8 cells, all RD-cGAs reach the same fitness value at the end of the execution. The goal is to analyse the behaviour of the RD-cGA variants when they obtain the same result.

On the basis of the fitness value displayed in Figs. 14, 15, 16 and 17, one can see that when addressing small-sized networks, all RD-cGA variants display the same behaviour. However, as the size of the network increases, the convergence rate, as well as the behaviour, of the RD-cGA variants starts to differ. In fact, as observed in Figs. 16-17, even if all the variants end up reaching the same result, they display a different way of reaching that result.

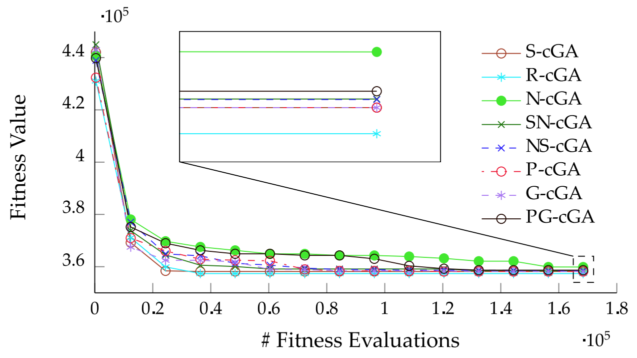

Fig. 17 Asynchronous RD-cGAs' fitness value evolution: network 2 of size 10 × 10 cells

Considering the results in Tables 1, 2, 3, 4, 5 and 6 and the fitness value evolution displayed in Figs. 10, 11, 12, 13, 14, 15, 16 and 17, some conclusions can be made about the effect of the ratio on the cGA. First, it is clear that the replacement (synchronous and asynchronous) has a huge influence on the type of adaptation needed. Second, it can be concluded that the efficiency of the ratio-control policy is mainly ruled by the size of the problem tackled. Actually, for small-sized problems, different policies with different behaviours and complexities have a greater probability to have the same efficiency, while as the size of the problem increases, the policies based on static ratios or dynamic ratios using a deterministic approach display more robustness than those based on dynamic approaches. Finally, in general, deterministic policies of adaptation (offline) proved to be as efficient as the online adaptation strategies.

Figure 18 represents a sample of the grid's radius value evolution for the asynchronous G-cGA, P-cGA and the PG-cGA when addressing Networks 1, 2 and 3 of size 10 × 10 cells. It is to be noted that this evolution is used to show the behaviour resulting from the ratio adaptation. Thus, it is not representative for all the executions or for its synchronous counterpart.

6.3 RD-cGAs versus state-of-the-art solvers

In order to prove that the ratio control is not merely an abstract idea, we extend our study to compare the best synchronous and asynchronous RD-cGA variants found in our study against some state-of-the-art algorithms, that have been previously proposed to solve the mobility management problem.

According to Tables 1, 2, 3, 4, 5 and 6, the G-cGA is the best synchronous variant, whereas the N-cGA is the best asynchronous variant. We make a further comparison

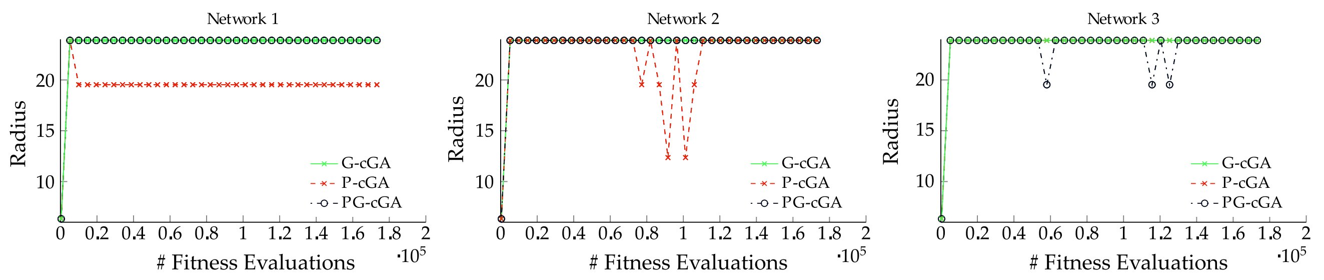

Fig. 18 Automatic ratio decisions of asynchronous G-cGA, P-cGA and PG-cGA when tackling networks of size 10×10 cells

Table 7 RD-cGAs vs SOTA: number of best solutions reached for networks with sizes 4×4, 6×6, 8×8 and 10×10 cells

Size	Network	NOMAD	CPLEX	GRASP	PBIL	GA	ABC	OI-GA	DE	N-cGA	G-cGA
4×4	1	**98535**	**98535**	**98535**	**98535**	**98535**	**98535**	**98535**	**98535**	**98535**	**98535**
	2	**97156**	**97156**	**97156**	97262	**97156**	**97156**	**97156**	**97156**	**97156**	**97156**
	3	**95038**	**95038**	**95038**	**95038**	**95038**	**95038**	**95038**	**95038**	**95038**	**95038**
6×6	1	177647	181677	175072	173804	176032	175041	**173701**	**173701**	**173701**	**173701**
	2	200069	200990	184142	**182331**	**182331**	**182331**	**182331**	**182331**	**182331**	**182331**
	3	175620	186481	175500	175564	176994	176148	**174519**	**174519**	**174519**	**174519**
8×8	1	316328	375103	316373	313336	312395	312558	311171	308401	**307695**	**307695**
	2	301833	351505	299616	292667	294391	297560	**287149**	**287149**	**287149**	**287149**
	3	271637	407457	270830	265853	265792	268366	**264204**	**264204**	**264204**	**264204**
10×10	1	404447	514504	402376	390489	391025	396456	387104	386681	386474	386351
	2	371091	468118	369979	362574	364354	366568	359623	358167	358431	358414
	3	382180	514514	383321	378033	378926	381725	372938	371829	371164	**370868**
# Optima		3	3	3	3	4	4	8	8	9	10

Best results are highlighted in bold

by considering both approximate and exact algorithms. For the approximate algorithms, we opt for the greedy randomized adaptive search procedure (GRASP), population-based incremental learning (PBIL), genetic algorithm (GA) and artificial bee colony (ABC), proposed in González-Álvarez et al. (2012). In addition, we consider the recently proposed oscillatory-increasing based genetic algorithm (OI-GA) devised in Dahi et al. (2016) and the differential evolution algorithm (DE) used in Almeida-Luz et al. (2011). For exact algorithms, we consider the IBM ILOG CPLEX solver and the nonlinear optimisation by mesh adaptive direct search (NOMAD), previously used in Berrocal-Plaza et al. (2014) to solve the RCP.

As it can be seen from Table 7, both RD-cGA variants outperform all state-of-the-art algorithms. The same assessment can be made on the basis of the metric "*Mean*" of Tables 8 and 9 . In fact, for Networks 1, 2 and 3 with sizes 4×4, 6×6 and 10×10 cells, Network 2 with size 8×8 cells, the N-cGA variant outperforms all other state-of-the-art solvers.

6.4 Statistical analysis

We perform here a series of statistical tests in order to enrich and ensure the actual validity of the found results presented all along Sect. 6. First, we start by conducting normality distribution and variance homogeneity tests using the Kolmogorov–Smirnov and Bartlett tests, respectively. Taking into account the results of the aforementioned tests, we have executed Friedman's test (see Table 10 and Figs. 19 and 20). Again, regarding the outcomes of the latter, we have performed the Tukey's honest significant difference as a post hoc.

One should bear in mind that all those tests have been conducted using a significance level of 5 % and this over several samples of data. Each sample represents the results obtained by a given algorithm throughout 30 executions. The statistical tests we have performed in this section have been made on both the static and dynamic RD-cGAs using the synchronous and asynchronous replacement strategies. However, no tests have been achieved to compare the best RD-cGAs against state-of-the-art algorithms. This is due to the fact that the results obtained by state-of-the-art solvers over the 30 executions are not available in the literature.

The results in Table 10 are the statistical counterpart of those in Tables 2, 3, 5 and 6 . They are organised as pairs $\{p$ value, $H_0\}$, where H_0 designates Friedman's null-hypothesis acceptance or rejection. The symbol "+" means that the null-hypothesis has been accepted, while "–" refers to the opposite. As it can be seen, the results of the synchronous RD-cGAs confirm those obtained when performing the ANOVA test in Alba and Dorronsoro (2005) (see Tables 3 and 4 of that work). Indeed, when using the synchronous replacement, no statistical difference can be found between the results of the static and dynamic (offline and online) ratio strategies.

Table 8 RD-cGAs vs SOTA: results obtained when tackling Networks 1, 2 and 3 with sizes 4 × 4 and 6 × 6 cells

	4×4 cells					6×6 cells				
Network	Algorithm	Best	Worst	Mean	Dev (%)	Algorithm	Best	Worst	Mean	Dev (%)
1	G-cGA	98535	98535	**98535.00**	0.00	G-cGA	173701	173701	**173701.00**	0.00
	N-cGA	98535	98535	**98535.00**	0.00	N-cGA	173701	173701	**173701.00**	0.00
	OI-GA	98535	98535	**98535.00**	-	OI-GA	173701	173701	**173701.00**	-
	CPLEX	98535	-	**98535.00**	0.00	CPLEX	181677	-	181677.00	0.00
	NOMAD	98535	-	100366.00	2.54	NOMAD	177647	-	189263.00	4.44
2	G-cGA	97156	97156	**97156.00**	0.00	G-cGA	182331	182331	**182331.00**	0.00
	N-cGA	97156	97156	**97156.00**	0.00	N-cGA	182331	182331	**182331.00**	0.00
	OI-GA	97156	97156	**97156.00**	-	OI-GA	182331	182331	**182331.00**	0.00
	CPLEX	97156	-	**97156.00**	0.00	CPLEX	200990	-	200990.00	0.00
	NOMAD	97156	-	100065.00	2.92	NOMAD	200069	-	221889.00	6.27
3	G-cGA	95038	95038	**95038.00**	0.00	G-cGA	174519	176847	174596.60	0.05
	N-cGA	95038	95038	**95038.00**	0.00	N-cGA	174519	174519	**174519.00**	0.00
	OI-GA	95038	95038	**95038.00**	-	OI-GA	174519	174519	**174519.00**	-
	CPLEX	95038	-	**95038.00**	0.00	CPLEX	186481	-	186481.00	0.00
	NOMAD	95038	-	100131.00	4.30	NOMAD	175620	-	182289.00	2.66

Best results are highlighted in bold

Table 9 RD-cGAs vs SOTA: results obtained when tackling Networks 1, 2 and 3 with sizes 8 × 8 and 10 × 10 cells

	8×8 cells					10×10 cells				
Network	Algorithm	Best	Worst	Mean	Dev (%)	Algorithm	Best	Worst	Mean	Dev (%)
1	G-cGA	307695	313585	311017.03	1.08	G-cGA	386474	400791	390363.13	1.01
	N-cGA	307695	313312	**310675.50**	0.97	N-cGA	386351	393267	**389233.10**	0.75
	OI-GA	311171	312925	312308.38	-	OI-GA	387104	394176	390531.47	-
	CPLEX	375103	-	375103.00	0.00	CPLEX	514504	-	514504.00	0.00
	NOMAD	316328	-	326008.00	2.19	NOMAD	404447	-	415740.00	1.82
2	G-cGA	287149	301393	290628.97	1.21	G-cGA	358431	367830	362417.97	1.11
	N-cGA	287149	297756	289105.50	0.68	N-cGA	358167	363339	**360722.33**	0.71
	OI-GA	287149	289771	**287236.41**	-	OI-GA	359623	370021	362320.88	-
	CPLEX	351505	-	351505.00	0.00	CPLEX	468118	-	468118.00	0.00
	NOMAD	301833	-	312497.00	2.59	NOMAD	371091	-	379725.00	1.31
3	G-cGA	264204	267041	265077.27	0.33	G-cGA	370868	381508	375757.23	1.32
	N-cGA	264204	266624	264903.10	0.26	N-cGA	371164	379987	**375346.17**	1.13
	OI-GA	264204	265324	**264533.16**	-	OI-GA	372938	382155	376900.28	-
	CPLEX	407457	-	407457.00	0.00	CPLEX	514514	-	514514.00	0.00
	NOMAD	271637	-	289309.00	3.09	NOMAD	382180	-	391627.00	0.81

Best results are highlighted in bold

Table 10 The results of Friedman's test

	Synchronous						Asynchronous					
Size \Network	I		II		III		I		II		III	
4×4 cells	1.0000	+	1.0000	+	1.0000	+	1.0000	+	1.0000	+	1.0000	+
6×6 cells	0.8957	+	1.0000	+	0.4289	+	0.0568	+	1.0000	+	0.8850	+
8×8 cells	0.1967	+	0.7978	+	0.1015	+	0.1667	+	0.2213	+	5.3980e-04	−
10×10 cells	0.0791	+	0.4315	+	0.3877	+	0.0110	−	0.0041	−	0.0021	−

So, this finding made here on the MMP confirms the one established in Alba and Dorronsoro (2005) concerning the maximum cut of graph problem (MaxCut). But this is not an always and globally valid rule. Surprisingly, in that same work, such behaviour could not be noticed on the minimum tardy task problem (MTTP), although the MMP, MaxCut and MTTP are all binary combinatorial problems. This gives two very valuable pieces of information. The first one is that, when using synchronous replacement, the factors influencing the efficiency of ratio-control policies go beyond the type of problem and the solution encoding and are more related to objective-function properties. This leads to the second finding, which is that it is highly probable that the MMP's features are similar to the ones of MaxCut in terms of fitness landscape and mapping between the solutions in the search space and fitness scenery.

Now, unlike the work done in Alba and Dorronsoro (2005) that studied only synchronous strategies, our research includes both synchronous and asynchronous policies, which allowed us to make new interesting discoveries. In fact, we found that when using the asynchronous replacement, the hypothesis, made in Alba and Dorronsoro (2005) and in our previous paragraph, does not hold. As a matter of fact, one can notice that even when tackling the same problem (i.e. MMP),

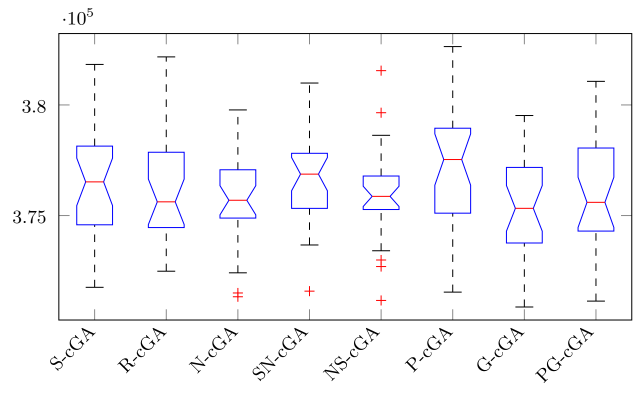

Fig. 19 Friedman's test on synchronous RD-cGAs

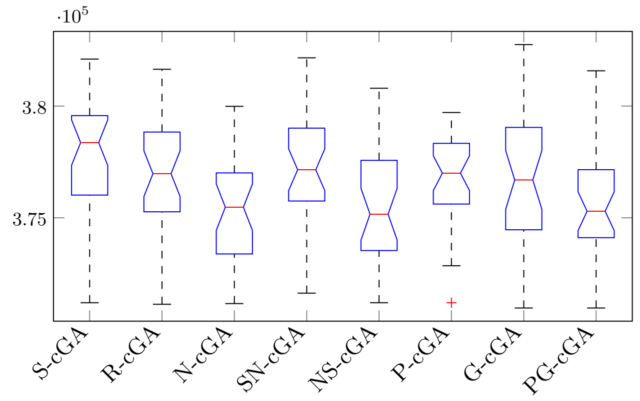

Fig. 20 Friedman's test on asynchronous RD-cGAs

a statistical difference starts to appear this time between the static and dynamic ratio strategies, which reflects the fact that they are not achieving the same results. Given these observations, the hypothesis that can be elaborated is that, unlike asynchronous replacement, the synchronous one makes the effect of ratio-adaptation strategies vanish.

So, considering all the results presented in Tables 2, 3, 5 and 6 , and the statistical ones in Table 10, one can state that the usefulness and outcome of a ratio adaptation depend on three main factors: the *problem being studied*, the *synchronism of replacement strategy* and the *ratio-adaptation policy*. Moreover, a profound knowledge of the problem being studied (e.g. fitness landscape) is necessary for a more suitable use and choice of the ratio-control strategy. Thus, neglecting one of these three factors might make the application of ratio-control strategies useless. For further understanding, additional materials on the statistical tests conducted here are also accessible at. [2]

[2] Tukey's results: https://tinyurl.com/Tukey-HSD.

7 The RD-cGAs: strengths, weaknesses and threats to validity

On the basis of the facts given all along this work and especially the numerical assessment in Sect. 6, one can notice that the studied RD-cGAs exhibit a number of advantages and disadvantages that made their strengths and weaknesses, respectively. Table 11 enumerates the principal ones that are, more or less, common to all the RD-cGA's variants investigated here.

Firstly, as can be seen, the RD-cGAs' use is not restricted to a specific type of problems, which makes them problem-independent and allows great flexibility in their application. Secondly, unlike hand-tuned (i.e. static) cGAs that have a constant ratio (i.e. grid structure) no matter the problem/instance being solved, the RD-cGAs contain an automatic mechanism that adapts the ratio to fit the problem's characteristics and therefore achieve better results. Furthermore, unlike most of the adaptation processes devised in the literature, the RD-cGAs' dynamic mechanism used to evolve the grid is computationally cheap (simple assignments and arithmetic operations). So, the RD-cGAs remain as simple as a hand-tuned cGA.

In addition to the above-mentioned facts, despite their simplicity, the RD-cGAs turn to outperform (or be competitive to) most of the state-of-the-art algorithms and this by achieving optimal or near-optimal solutions. These solutions are obtained after a reasonable period of time which is ideal for real-life problems that generally require real-time solving. Moreover, such efficiency is achieved with very rudimentary and basic settings of the RD-cGAs parameters, which ease their use even by inexperienced users. Besides, the RD-cGAs turn to be scalable and this by maintaining their low complexity and efficiency even when solving problems of different sizes. This can be very helpful in real-life scenarios where the problems have generally different sizes. Finally, since the cGA (i.e. skeleton of the RD-cGAs) is one the most efficient and widely spread sEAs, an extensive literature exists on it. This appears to be useful for the future users when trying to understand and control the RD-cGAs under given circumstances or scenarios. Now, this being said, despite showing many advantages, the RD-cGAs have also disadvantages. The latter consist in introducing additional tunable parameters such as the value of ϵ in Eqs. (5)–(12) and the adaptation condition in Sect. 4.2.2.

Now, for the sake of more valuable, helpful and reliable research, we go further by performing a thorough theoretical analysis in order to extract possible threats to the validity of our work, and we also suggest how to cope with them. For this purpose, we have rigorously analysed our work according to four well-established and widely used kinds of validity threats criteria (De Oliveira Barros and Dias-Neto 2011; Malhotra and Khanna 2018; Zhang et al. 2018): *conclusion*, *inter-*

Table 11 Advantages and disadvantages of the RD-cGAs

Advantages	Disadvantages
Can be applied to solve a wide range of problems	Introduce extra tunable parameters
Automatically adapt the grid's structure to fit the problem's features	
Use a computationally-cheap grid-adaptation mechanism	
Have a low complexity equal to the one of hand-tuned cGA	
Outperform/as efficient as most of the state-of-the-art algorithms	
Achieve optimal/suboptimal solutions in reasonable time	
Attain optimal/suboptimal solutions using basic parameters' settings	
They are scalable when solving problems of different sizes	
A rich literature exists on the cGA which ease the RD-cGAs' understanding and control	

Table 12 The RD-cGAs: threats to validity

Conclusion Validity Threats	Not accounting for random variation	No
	Lack of good descriptive statistics	No
	Lack of a meaningful comparison baseline	No
	Lack of formal hypothesis and statistical tests	No
Internal Validity Threats	Poor parameter settings	No
	Lack of clear data collection tools and procedures	No
	Lack of real problem instances	No
Construct Validity Threats	Lack of assessing the validity of cost measures	No
	Lack of assessing the validity of effectiveness measures	No
	Lack of discussing the underlying model subjected to optimisation	No
External Validity Threats	Lack of a clear definition of target instances	Yes
	Lack of a clear object selection strategy	No
	Lack of evaluations for instances of growing size and complexity	No

nal, *construct* and *external* validities (see Table 12). It is to be noted that "No" means that our study does not suffer from that threat, while "Yes" designates the alternative scenario.

The analysis shows that the work we conduct here can potentially suffer only from one minor and almost-unavoidable *subkind* of *external* validity threats: "*lack of a clear definition of target instances*". This refers to the fact that the generalisation of the obtained results to other problems becomes tricky if there is no enough comprehension of the problem instances used during the study (De Oliveira Barros and Dias-Neto 2011). In our case, this issue is due to two main reasons. The first one is, simply, that in order to cope with such threat, sufficient knowledge on the studied problem has to be available, which is not the case for the MMP. Actually, as far as the authors know, no information on its features can be found in the literature and discovering them goes beyond the scope of our work knowing that substantial and sophisticated fitness landscape and search space analysis theory lies behind that.

The second cause for the above-mentioned threat goes to the fact that even with enough information on the MMP's properties, there is no guarantee that the results we obtain here when solving the MMP can be generalised to other problems. This is logical and very likely to happen in studies implying the use of metaheuristics for solving optimisation problems. Indeed, in order that the findings obtained on a given problem can (but not surely) be generalised to all other existing problems in the world, many complex and hard-to-satisfy constraints have to be fulfilled. The principal one is that the studied problem itself has to represent all the properties (e.g. complexity, multimodality, etc.) contained in those existing optimisation problems. However, such thing is impossible considering that every optimisation problem has its own characteristics and it is hardly feasible for one single problem to represent all of them. Therefore, coping with such an issue will probably imply performing experiments on a wide range of problems that represents, all together, a large set of problems' attributes. Nonetheless, as it can be thought, for numerous reasons, this is a many-works effort and goes beyond the scope of one paper. Indeed, trying to do this in one paper, like it was previously done in Alba and Dorronsoro (2005), will imply reproducing the same issues of that work and this by neglecting other very important aspects that need to be studied and, therefore, make the study suffers from more important and severe threats (e.g. construct, conclusion and internal threats) than the external ones.

The goal of our work is to set the basis for a more reliable and flawless study on which further theory and studies can be built on. Technically speaking, we have corrected the major validity flaws in Alba and Dorronsoro (2005), but for almost-unavoidable threats like the external ones, as a solution, one might go for a systematic and profound analysis like the one we conduct here, but using another validation problem that represents other characteristics than the ones illustrated by the MMP. Once this is done, one can include a qualitative comparison between the obtained results and ours.

8 Conclusion and future work

In this paper, we answered some of the most interesting questions that were pending and surrounding the relationship between the ratio and designing better cGAs. As a matter of fact, it is rather clear now that the ratio adaptation has a direct influence on the search process of the cGAs and this by enhancing or decreasing their efficiency and scalability. The outcome of the ratio adaptation can be controlled by an adequate choice of the strategy used and the replacement synchronism. The latter has to be in convenience with the size and complexity of the problem. As general but not absolute conclusion, simple deterministic then secondly dynamic strategies using asynchronous replacement are more advised when engaging real-world problems such as the MMP.

For future research lines, we are considering exploring cooperation strategies and island-based models between several policies of ratio-control. We also intend to design evolving ratio-control strategies that can switch between several types of strategies. Finally, we aim at extending our study to larger instances and compare them to other state-of-the-art optimisers.

Compliance with ethical standards

Conflict of interest The authors would like to thank Mr. Louay Rabah Dahi for his help in checking the numerical results. Also, author B acknowledges partial funding from Spanish-plus-FEDER and MINECO project moveON TIN2014-57341-R, TIN2016-81766-REDT and TIN2017-88213-R. Author A declares that he has no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

(2008) Mobility management problem benchmark instances. http://oplink.lcc.uma.es/problems/mmp.html

Al-Naqi A, Erdogan AT, Arslan T, Mathieu Y (2010) Balancing exploration and exploitation in an adaptive three-dimensional cellular genetic algorithm via a probabilistic selection operator. In: Proceedings of the NASA/ESA conference on adaptive hardware and systems. pp 258–264

Al-Naqi A, Erdogan AT, Arslan T (2011) Fault tolerant three-dimensional cellular genetic algorithms with adaptive migration schemes. In: Proceedings of the NASA/ESA conference on adaptive hardware and systems. pp 352–359

Alba E, Dorronsoro B (2005) The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. IEEE Trans Evol Comput 9:126–142

Alba E, Dorronsoro B (2008) Cellular genetic algorithms, 1st edn. Springer, Berlin

Alba E, Troya JM (2000) Cellular evolutionary algorithms: evaluating the influence of ratio. In: Proceedings of the 6th international conference on parallel problem solving from nature, (PPSN VI). Springer, pp 29–38

Almeida-Luz SM, Vega-Rodriguez MA, Gomez-Pulido JA, Sanchez-Perez JM (2011) Differential evolution for solving the mobile location management. Appl Soft Comput 11(1):410–427

Bäck T (1993) Optimal mutation rates in genetic search. In: Proceedings of the 5th international conference on genetic algorithms. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 2–8

Banharnsakun A (2019) Artificial bee colony algorithm for solving the knight's tour problem. In: Vasant P, Zelinka I, Weber GW (eds) Proceedings of the international conference on intelligent computing and optimization (ICO 2018). Springer International Publishing, pp 129–138

Bar-Noy A, Kessler I (1993) Tracking mobile users in wireless communications networks. IEEE Trans Inf Theory 39(6):1877–1886

Berrocal-Plaza V, Vega-Rodriguez MA, Sanchez-Perez JM (2014) A strength pareto approach to solve the reporting cells planning problem. In: Proceedings of the 14th international conference on computational science and its applications, (ICCSA). Springer, vol 8584, pp 212–223

Dahi ZA (2017) Optimisation problem solving in the field of cellular networks. PhD thesis, Constantine 2 University

Dahi ZA, Mezioud C, Alba E (2016) A novel adaptive genetic algorithm for mobility management in cellular networks. In: Proceedings of the 11th international conference on hybrid artificial intelligent systems, (HAIS). Springer, pp 225–237

Dahi ZA, Alba E, Draa A (2018) A stop-and-start adaptive cellular genetic algorithm for mobility management of GSM-LTE cellular network users. Expert Syst Appl 106:290–304

De Oliveira Barros M, Dias-Neto AC (2011) Threats to validity in search-based software engineering empirical studies, pp 1–12. UNIRO

Dorronsoro B, Bouvry P (2011) Adaptive neighborhoods for cellular genetic algorithms. In: Proceedings of the IEEE international symposium on parallel and distributed processing workshops and Phd forum. pp 388–394

Eiben AE, Smith JE (2015) Parameters and parameter tuning. Springer, Berlin Heidelberg, pp 119–129

Fahad AM, Ahmed AA, Kahar MNM (2019) Network intrusion detection framework based on whale swarm algorithm and artificial neural network in cloud computing. In: Vasant P, Zelinka I, Weber GW (eds) Proceedings of the international conference on intelligent computing and optimization (ICO 2018). Springer International Publishing, pp 56–65

González-Álvarez DL, Rubio-Largo A, Vega-Rodríguez MA, Almeida-Luz SM, Gómez-Pulido JA, Sánchez-Pérez JM (2012) Solving the reporting cells problem by using a parallel team of evolutionary algorithms. Logic J IGPL 20(4):722–731

Grefenstette JJ (1986) Optimization of control parameters for genetic algorithms. IEEE Trans Syst Man Cybern 16(1):122–128

Hać A, Zhou X (1997) Locating strategies for personal communication networks, a novel tracking strategy. IEEE J Sel Areas Commun 15(8):1425–1436

Huang A, Li D, Hou J, Bi T (2015) An adaptive cellular genetic algorithm based on selection strategy for test sheet generation. Int J Hybrid Inf Technol 8:33–42

Jie L, Liu W, Sun Z, Teng S (2017) Hybrid fuzzy clustering methods based on improved self-adaptive cellular genetic algorithm and optimal-selection-based fuzzy c-means. Neurocomputing 249:140–156

Kamkar I, Akbarzadeh TM (2010) Multiobjective cellular genetic algorithm with adaptive fuzzy fitness granulation. In: Proceedings of the IEEE international conference on systems, man and cybernetics. pp 4147–4153

Karafotias G, Hoogendoorn M, Eiben AE (2015) Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans Evol Comput 19(2):167–187

Lechuga GP, Sánchez FM (2019) Modeling and optimization of flexible manufacturing systems: A stochastic approach. In: Vasant P, Zelinka I, Weber GW (eds) Proceedings of the international conference on intelligent computing and optimization (ICO 2018). Springer International Publishing, pp 539–546

Lin L, Gen M (2009) Auto-tuning strategy for evolutionary algorithms: balancing between exploration and exploitation. Soft Comput 13(2):157–168

Malhotra R, Khanna M (2018) Threats to validity in search-based predictive modelling for software engineering. IET Softw 12(4):293–305

Morales-Reyes A, Stefatos EF, Erdogan AT, Arslan T (2008) Towards fault-tolerant systems based on adaptive cellular genetic algorithms. In: Proceedings of the NASA/ESA conference on adaptive hardware and systems. pp 398–405

Pang J, He J, Dong H (2018) Hybrid evolutionary programming using adaptive lévy mutation and modified nelder-mead method. Soft Comput. https://doi.org/10.1007/s00500-018-3422-4

Razavi S (2011) Tracking area planning in cellular networks. PhD thesis, Department of Science and Technology, Linkoping University

Sarma J, De Jong K (1996) An analysis of the effects of neighborhood size and shape on local selection algorithms. In: Proceedings of the 4th international conference on parallel problem solving from nature parallel problem solving from nature, (PPSN IV). Springer, pp 236–244

Sarma J, Jong KAD (1997) An analysis of local selection algorithms in a spatially structured evolutionary algorithm. In: Proceedings of the 7th international conference on genetic algorithms. pp 181–187

Schraudolph NN, Belew RK (1992) Dynamic parameter encoding for genetic algorithms. Mach Learn 9(1):9–21

Sivanandam SN, Deepa SN (2007) Introduction to genetic algorithms, 1st edn. Springer, Berlin

Sun CT, Wu MD (1995) Self-adaptive genetic algorithm learning in game playing. In: Proceedings of the IEEE international conference on evolutionary computation. vol 2, pp 814–818

Talbi EG (2009) Metaheuristics: from design to implementation. Wiley Publishing, Hoboken

Tian M, Gao X (2019) Differential evolution with neighborhood-based adaptive evolution mechanism for numerical optimization. Inf Sci 478:422–448

Torres-Escobar R, Marmolejo-Saucedo JA, Litvinchev I, Vasant P (2019) Monkey algorithm for packing circles with binary variables. In: Vasant P, Zelinka I, Weber GW (eds) Proceedings of the international conference on intelligent computing and optimization (ICO 2018). Springer International Publishing, pp 547–559

Zhang J, Chen WN, Zhan ZH, Yu WJ, Li YL, Chen N, Zhou Q (2012) A survey on algorithm adaptation in evolutionary computation. Front Electr Electron Eng 7(1):16–31

Zhang L, Tian JH, Jiang J, Liu YJ, Pu MY, Yue T (2018) Empirical research in software engineering-a literature survey. J Comput Sci Technol 33(5):876